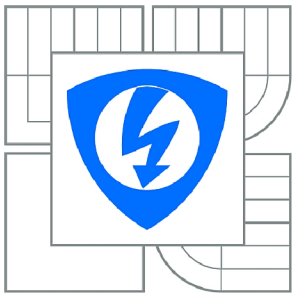
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

TESTOVACÍ IMPLEMENTACE PROTOKOLU ACP

TEST IMPLEMENTATION OF THE ACP PROTOCOL

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

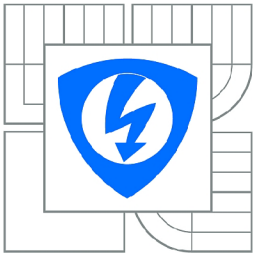
AUTOR PRÁCE — Bc. PETR LEŽÁK

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. KAREL BURDA, CSc.

SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Petr Ležák ***ID:*** 106225
Ročník: 2 ***Akademický rok:*** 2011/2012

NÁZEV TÉMATU:

Testovací implementace protokolu ACP

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte a popište univerzální protokol řízení přístupu ACP (Access Control Protocol). Navrhněte a zdůvodněte koncept implementace tohoto protokolu pro účely jeho testování. Pro testování protokolu rovněž navrhněte několik scénářů řízení přístupu. Na tomto základě testovací implementaci protokolu ACP naprogramujte, ověřte a zhodnoťte. Testovací implementace by měla být otevřená, tj. měla by umožnit definovat a testovat další možné scénáře řízení přístupu. Pro vytvořený program zpracujte návod k jeho obsluze.

DOPORUČENÁ LITERATURA:

[1] Burda, K.: Univerzální rámec pro řízení přístupu v počítačových sítích. Elektrorevue 2011/9. 6 s.
[2] Sharp, R.: Principles of Protocol Design. Springer. Berlin 2008.

Termín zadání: 6.2.2012 ***Termín odevzdání:*** 24.5.2012

Vedoucí práce: doc. Ing. Karel Burda, CSc.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

Diplomová práce obecně pojednává o metodách řízení přístupu, jednotlivých jeho modulech a konkrétně pak o autentizaci žadatelů. Jsou zde uvedeny metody autentizace využitelné v testovací implementaci protokolu ACP. Dále je rozebrán protokol ACP, jeho možnosti a způsoby použití. Podrobně je popsán formát ACP zpráv, formát i typy AVP a mechanismus transakcí protokolu ACP.
Vlastní práce je pak zaměřena na návrh software pro testování tohoto protokolu. Jsou zde rozebrány možnosti testování tohoto protokolu a navrženy testovací scénáře. Následně jsou sepsány požadavky na testovací software a navržen způsob jeho realizace.
Dále je v práci uvedena technická dokumentace k vytvořenému programu. V ní jsou vysvětleny hlavní myšlenky, které jsou v programu použity, popsán účel jednotlivých částí programu a vazby mezi nimi.
Nakonec je v příloze zpracován návod k obsluze programu včetně názorného příkladu popisujícího naprogramování a otestování jednoduchého scénáře autentizace.

KLÍČOVÁ SLOVA

ACP, AAA, řízení přístupu, autentizace, testování

ABSTRACT

In general this master's thesis deals with access control methods and their individual modules and in particular with authentication of supplicants. There are listed authentication methods useful in the implementation of the ACP protocol. ACP protocol is also discussed including possibilities and uses. ACP message format is described in detail with AVP format and types. The transaction mechanism is also mentioned here.
The main part of the thesis is focused on software design for protocol testing. Possibilities of the testing are discussed and test scenarios are suggested. Consequently, requirements for test software are listed and its implementation is designed.
Furthermore, there is technical documentation of the program. The main ideas used in the program are explained in it. The purpose of each part of the program is written including links between them.
Finally, there is a manual for the program. It also contains an illustrative example describing how to make and test a simple scenario of the authentication.

KEYWORDS

ACP, AAA, access control, authentication, test

LEŽÁK, Petr *Testovací implementace protokolu ACP*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 67 s. Vedoucí práce byl doc. Ing. Karel Burda, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Testovací implementace protokolu ACP“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

Děkuji panu doc. Ing. Karlovi Burdovi, CSc.
za odborné vedení práce.
Děkuji manželce Bc. Daně Ležákové za
velkou trpělivost a podporu.

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

V kapitole 1.1 je obecně rozebrána problematika řízení přístupu. Je zde vysvětlena funkce jednotlivých modulů, které řízení přístupu zajišťují. Podrobněji je v kapitole 1.2 popsána autentizace, protože je nutná pro návrh scénářů protokolu ACP.

Kapitola 1.3 detailně popisuje protokol ACP. Je zde uveden příklad nasazení tohoto protokolu ve firmě a na něm vysvětleny jeho možnosti. Podrobně je rozpracována struktura ACP zpráv v sekci 1.3.1 a v nich obsažených AVP v sekci 1.3.2. Dále je v sekci 1.3.3 uveden přehled typů AVP a vysvětleno jejich použití. V sekci 1.3.4 je pak rozebrán mechanismus tvorby transakcí a způsob předávání zpráv v rámci nich přes více portálů ACP.

Kapitola 1.4 popisuje testování protokolu ACP. Jsou zde sepsány testované vlastnosti a pak navrženy testovací scénáře běhu protokolu. Na základě toho jsou v kapitole 1.5 uvedeny požadované vlastnosti testofacího softwaru. Dále je v sekci 1.5.1 navržena testovací struktura ACP sítě a v kapitole 1.6 je navržena struktura programu.

Kapitola 1.7 obsahuje popis jednotlivých částí programu a vysvětlení vztahů mezi nimi. Jsou zde popsány hlavní koncepty využité při tvorbě programu. Případný zájemce o rozšiřování programu tak bude vědět, jaké úpravy a kde má dělat.

1 ŘEŠENÍ DIPLOMOVÉ PRÁCE

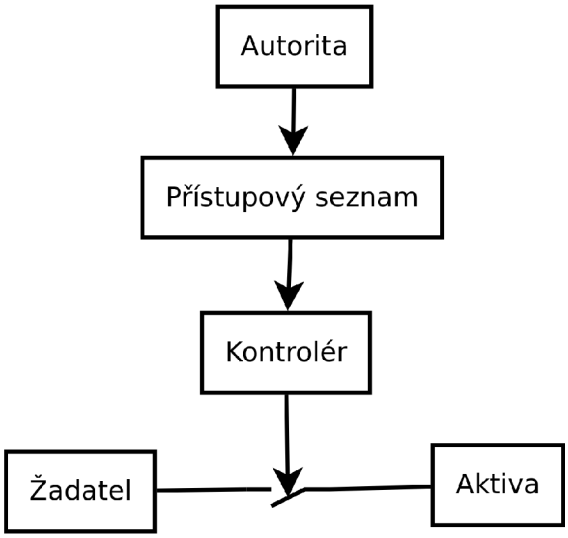
1.1 Řízení přístupu

Podle [1] je řízení řístupu bezpečnostní opatření umožňující přístup k aktivům jeho uživatelům a bránící v přístupu případným útočníkům. Přístup může být buď místní nebo dálkový.

Místní přístup k aktivům je například čtení dokumentů v archivu, přístup k pracovní stanici nebo serveru. K řízení se využívají zejména prostředky fyzické bezpečnosti, jako například bezpečnostní dveře a trezory. Tato problematika je nad rámec této práce.

Dálkový přístup umožňuje přístup k aktivům pomocí telekomunikační sítě (nejen počítačové, ale i například telefonní). Příkladem je přístup do e-mailové schránky nebo služba telefonního bankovnictví.

Obecné schéma řízení přístupu je vidět na obr. 1.1.



Obr. 1.1: Řízení přístupu

O přístupu žadatele k aktivům rozhoduje kontrolér - na základě identifikačních údajů žadatele a přístupového seznamu provádí autorizaci. Přístupový seznam obsahuje identifikační údaje všech uživatelů a přístupová práva těchto uživatelů k aktivům. O obsahu přístupového seznamu rozhoduje autorita - osoba nebo organizace spravující daná aktiva.

Aby žadatel získal přístup k aktivům, tak musí předat kontroléru své identifikační údaje (například uživatelské jméno) - provede identifikaci. Kontrolér pak prověří, zda je uživatel opravdu tím, za koho se vydává - provede autentizaci - a podle přístupového seznamu ověří, zda má žadatel přístup k požadovaným aktivům.

1.2 Autentizace

Identitu žadatele lze ověřit několika různými způsoby. Toto popisují třídy autentizací, jak je popsáno v [1]:

a) autentizace znalostí - žadatel svoji identitu prokáže znalostí tejné informace (hesla, PINu, odpovědi na předem dohodnutou otázku atd.).

b) autentizace předmětem - žadatel svoji identitu prokáže vlastnictvím určitého předmětu. Při autentizaci na dálku není možné ověřit fyzické vlastnictví předmětu, kontroluje se znalost informace uložené v předmětu - jedná se tedy opět o autentizaci znalostí, jen nositelem znalosti není člověk, ale předmět. Příkladem předmětu je čipová karta, token generující časově proměnná hesla, ale i například certifikát uložený na počítači žadatele. Z posledního uvedeného příkladu je vidět, že předmět nemusí mít fyzickou podobu.

c) autentizace žadatelem - žadatel se prokazuje svými biometrickými údaji (otisk prstu, snímek obličeje, ...).

Žádná z výše uvedených metod není absolutně bezpečná. Znalost (heslo, PIN) je možné zachytit monitorem klávesnice, předmět je možné odcizit a charakteristiky žadatele je možné bez jeho vědomí sejmout a vyrobit například umělý prst s jeho otiskem. Proto se výše uvedené třídy autentizace často kombinují. Například k výběru peněz z bankomatu je třeba čipová karta a PIN, jedná se tedy o autentizaci předmětem a znalostí.

Dále popíši autentizace, které připadají v úvahu využít pro testovací implementaci protokolu ACP.

1.2.1 Autentizace jednorázovým heslem

Žadatel prokáže svou identitu zasláním hesla kontroléru. Heslo lze využít pouze jednou, odpadá tedy riziko odposlechu hesla útočníkem. Uživatel obdrží seznam jednorázových hesel a stejný seznam je uložen v přístupovém seznamu. Pro zvýšení bezpečnosti lze uložit jen seznam otisků hesel. Po použití hesla ho kontrolér ze seznamu vyřadí.

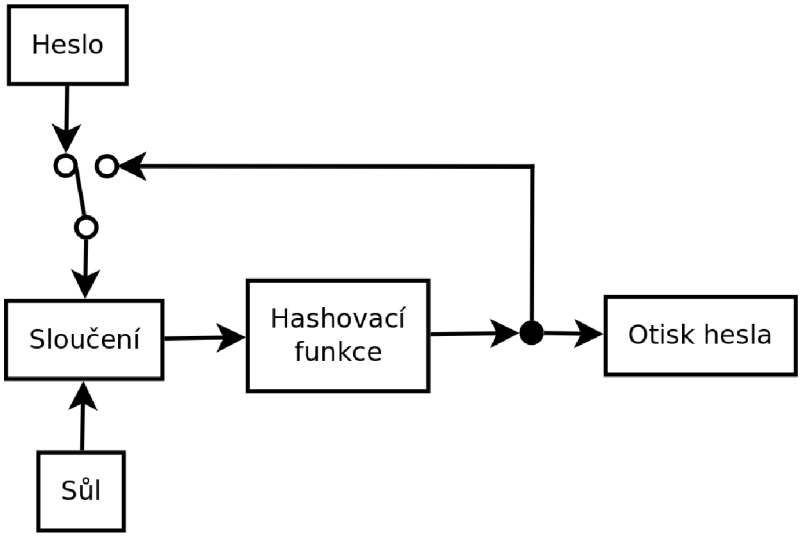
1.2.2 Autentizace časově proměnným heslem

Časově proměnná hesla jsou generována pomocí tokenů. V tokenu je uložený tajný klíč K, stejný klíč je uložený v přístupovém seznamu jako hodnota K'. Žadatel vypočítá jednorázové heslo podle vztahu $h_t = H(t, K)$, kde t je čas. Hashovací funkce H zajistí, že útočník nedokáže ze zaslaného hesla vypočítat tajný klíč K. Kontrolér vypočítá hodnotu $h'_t = H(t, K')$ a porovná ji s heslem h_t získaným od žadatele. Pokud se hesla budou shodovat, tak kontrolér přijme autentizaci.

1.2.3 Autentizace stálým heslem

Žadatel svoji identitu prokáže zasláním hesla kontroléru. Heslem je zde obecně myšlena jakákoli informace - klasická hesla, PINy, odpovědi na kontrolní otázky a podobně. Heslo musí být dostatečně bezpečné, aby je útočník nemohl uhodnout. Útočník může k uhodnutí hesla použít útok hrubou silou (zkoušením všech možných kombinací znaků) nebo slovníkový útok (zkoušením slov ze slovníku). Proto, aby heslo bylo bezpečné, tak musí být dostatečně dlouhé, aby počet možných kombinací hesla byl velký, a musí se jednat o náhodnou kombinaci písmen a číslic, aby nešel použít slovníkový útok.

Informace o heslech musí být uloženy v přístupovém seznamu, aby kontrolér mohl ověřit heslo zadané žadatelem. Uložit heslo přímo ale není bezpečné, protože přístupový seznam může být odcizen a útočník by pak měl do systému přístup. Proto se neukládá heslo, ale jeho otisk, jak je vidět na obr. 1.2.



Obr. 1.2: Autentizace stálým heslem

Při ukládání hesla do přístupového seznamu se nejdříve vygeneruje náhodná posloupnost bytů, tzv. sůl. Heslo je pak sloučeno se solí například tak, že se sůl připojí za něj, a na výslednou posloupnost bytů se aplikuje hashovací funkce (MD5, SHA, ...). Celý proces je možné mnohokrát opakovat, aby se prodloužila doba ověření hesla pro útočníka. Sůl zajistí, že stejné heslo bude mít vždy jiný otisk. Útočník pak neuvidí, kteří uživatelé mají stejné heslo, navíc si nebude moci předpočítat slovník otisků různých hesel, protože počet kombinací hesla+soli bude mnohem vyšší než počet kombinací samotného hesla. Do přístupového seznamu je uložen otisk hesla, použitá sůl a počet iterací výpočtu otisku. Bude-li kontrolér chtít ověřit heslo zadané uživatelem, tak výše popsaný výpočet provede s uživatelem zadaným heslem a solí

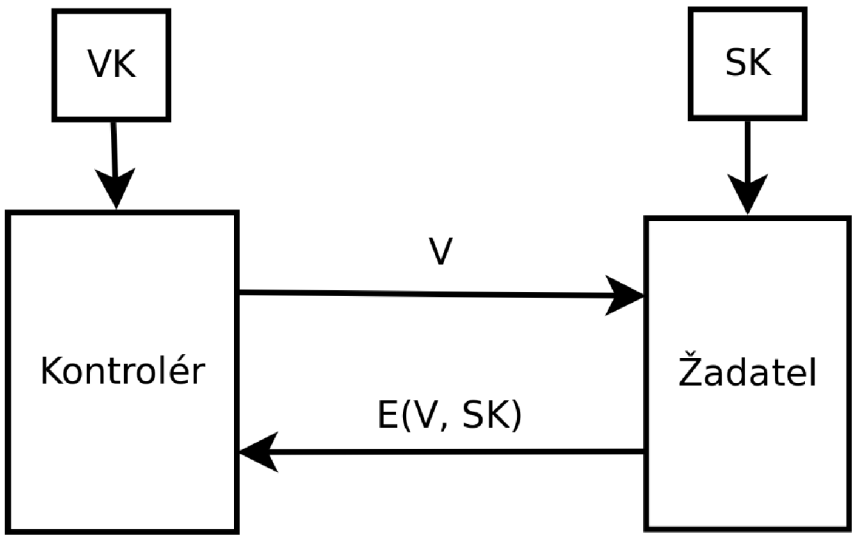
načtenou z přístupového seznamu. Shodují-li se hesla, pak se musí shodovat i jejich otisky.

1.2.4 Autentizace symetrickou šifrou

Jedná se o autentizaci předmětem. Žadatel i kontrolér mají k dispozici stejný tajný klič K. Kontrolér vygeneruje náhodnou výzvu V, kterou odešle žedateli. Žadatel výzvu zašifruje a kryptogram $C = E(V, K)$ odešle kontroléru. Kontrolér kryptogram dešifruje a pokud platí $D(C, K) = V$, tak autentizaci přijme. Nevýhodou použití symetrické šifry je nutnost mít jedinečný klíč pro každou dvojici žadatel-kontrolér.

1.2.5 Autentizace asymetrickou šifrou

Autentizace asymetrickou šifrou odstraňuje nevýhody autentizace symetrickou šifrou. Její průběh je vidět na obr. 1.3. Žadatel má k dispozici svůj soukromý klíč SK a jemu odpovídající veřejný klíč VK. Veřejný klíč VK uloží autorita do přístupového seznamu, soukromý klíč SK si žadatel udrží v tajnosti (v zašifrovaném souboru na počítači nebo na čipové kartě). Kontrolér vygeneruje výzvu V, tedy náhodnou posloupnost bytů, a odešle ji žadateli. Žadatel k výzvě může přidat další informace (například identifikaci kontroléru), zašifruje ji (tj. podepíše) svým soukromým klíčem SK a odešle kontroléru. Ten pomocí veřejného klíče VK vyzvednutého z přístupového seznamu ověří platnost podpisu a podepsaných údajů a ověří tak znalost soukromého klíče žadatelem. Výhodou tohoto způsobu autentizace je, že žadatel může bezpečně využívat jeden pár klíčů pro autentizaci na více kontrolérech.



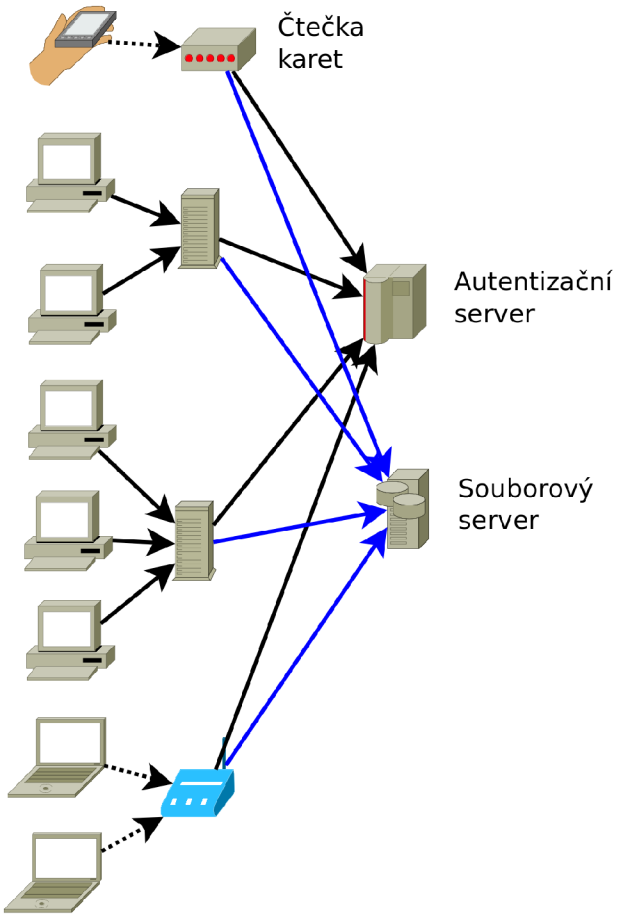
Obr. 1.3: Autentizace asymetrickou šifrou

Autentizace asymetrickou šifrou se využívá například v protokolu SSH (secure shell). Její implementace je popsána v [2]. Výzvou je zde hodnota *session identifier*, která je vypočítána při výměně klíčů relace, soukromý klíč uživatele je uložen v souboru id_rsa, odpovídající veřejný klíč v souboru id_rsa.pub a seznam veřejných klíčů, jejichž vlastníci se mohou přihlásit (tedy přístupový seznam), v souboru authorized_keys. Všechny soubory jsou umístěny v adresáři .ssh/ v domovském adresáři uživatele.

1.3 Protokol ACP

Informace o protokolu ACP jsem čerpal z [3, 4, 5].

Access Control Protocol (ACP) je protokol pro řízení přístupu k aktivům vyvinutý na Vysokém učení technickém v Brně. Protokol není závislý na použité přenosové vrstvě a ani ji nijak nespecifikuje. Komplexně řeší řízení přístupu ve velkých organizacích. Vzorový příklad ACP sítě ve firmě je vidět na obr. 1.4.



Obr. 1.4: Příklad ACP sítě ve firmě

Centrálním prvkem této vzorové sítě je autentizační server, který spravuje identifikační údaje všech uživatelů. Dále je tu několik serverů s různými aktivy, například souborový server. Dále jsou tu brány, které zajišťují konverzi transportní vrstvy a také koncentraci datových toků z různých zdrojů. Nejníže v hierarchii jsou klientské počítače a jiná zařízení (například čtečka karet u vstupních dveří).

Protokol ACP je transakčně orientovaný. Aby žadatel získal přístup k aktivu, tak musí vytvořit transakci, identifikovat a autentizovat se a specifikovat požadované aktivum. Kontrolér rozhodne, zda je požadavek oprávněný, a pokud ano, tak požadované aktivum žadateli odešle a transakci ukončí.

Jedna transakce je tedy platná mezi dvěma počítači - iniciátorem a adresátem. Často je ale nutné, aby žadatel komunikoval s více počítači. Například autentizační server žadatele autentizuje a souborový server poskytne požadované aktivum. Protokol ACP poskytuje několik možností jak toto řešit, viz [4]:

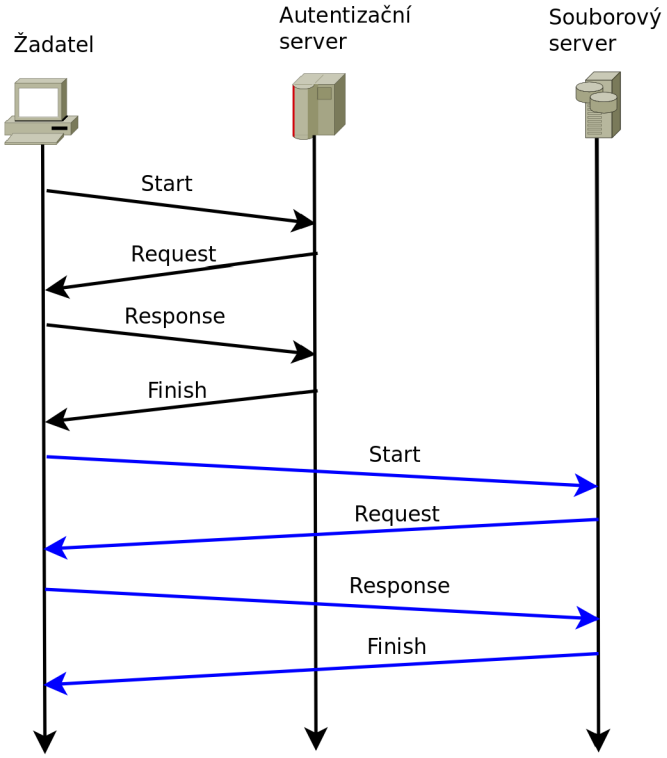
Při sekvenčním přístupu žadatel vytváří transakce postupně, následující transakci zahájí až po úspěšném dokončení předchozí. Na obr. 1.5 je vidět výše uvedený příklad s přístupem k souborovému serveru. Žadatel nejdříve vytvoří transakci s autentizačním serverem, autentizuje se a obdží dočasně platný certifikát obsahující jeho identitu a přístupová práva podepsaný autentizačním serverem. Poté vytvoří transakci se souborovým serverem, autentizuje se tímto certifikátem a obdrží požadované aktivum - soubor. Získaný certifikát může dále využívat pro svou autentizaci vůči jiným serverům v ACP síti.

Jinak pracuje přístup vložením jedné transakce do druhé. Ten spočívá v tranzitování AVP mezi více servery. Na obr. 1.6 je vidět, jak tento přístup pracuje ve výše uvedeném případě přístupu k souborovému serveru. Žadatel nejdříve vytvoří transakci se souborovým serverem a specifikuje požadované aktivum. Souborový server vytvoří transakci s autentizačním serverem a ve zprávě Start specifikuje třidu aktiv, které žadatel požaduje. Autentizační server následně zahájí autentizaci a souborový server předává příslušná AVP mezi žadatelem a autentizačním serverem. Pokud autentizační server ve zprávě Finish autentizaci přijme, tak souborový server odešle žadateli požadované aktivum. Podrobný popis lze nalézt v [4].

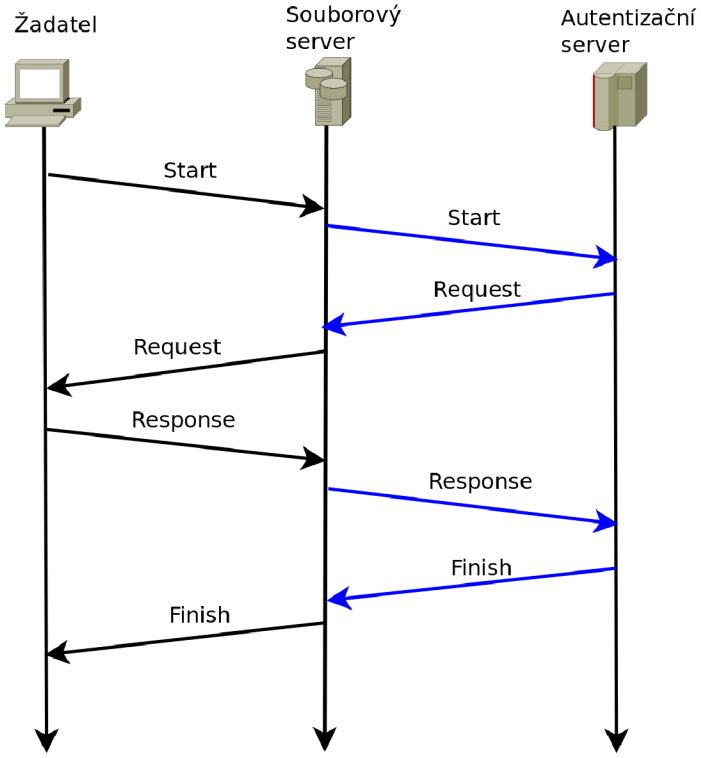
1.3.1 Formát zprávy

Protokol ACP přenáší svá data pomocí ACP zpráv, jejichž formát je definován v [3]. Zpráva se skládá z hlavičky následované několika AVP. Hlavička obsahuje informace o typu zprávy a její příslušnosti k určité transakci. Jednotlivá AVP obsahují užitečné informace přenášené zprávou. Struktura zprávy je vidět na obr. 1.7.

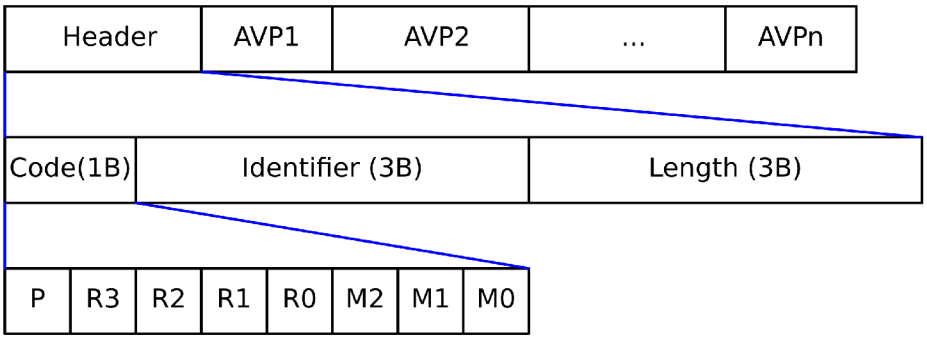
Hlavička zprávy obsahuje pole *Code* o velikosti 8 bitů. Nejvíce významný bit *P* slouží k odlišení protokolu ACP od protokolu EAP, musí být nastaven na 1. Následují



Obr. 1.5: Sekvenční zřetězení transakci



Obr. 1.6: Vkládání transakcí



Obr. 1.7: Formát ACP zprávy

4 rezervní bity *R3-R0*, ty musí být nastaveny na 0. Následující 3 bity *M2-M0* určují typ zprávy.

Dále hlavička obsahuje identifikátor transakce *Identifier* o velikosti 3 byty. *Identifier* jednoznačně přiřazuje zprávu k transakci iniciované jedním směrem. Abychom mohli úplně jednoznačně identifikovat transakci, tak hodnotu *Identifier* musíme zkombinovat s bitem *M0*, který určuje zda zpráva míří od Iniciátora k Adresátovi nebo naopak. Tento mechanismus je blíže vysvětlen v sekci 1.3.4.

Pole *Length* udává celkovou délku zprávy v bytech. Do této délky je zahrnuta i hlavička.

Protokol ACP má definované následující typy zpráv:

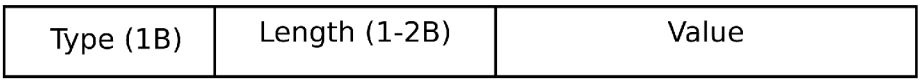
- Start - zahájení transakce
- Offer - adresát dává žadateli nabídku dostupných aktiv nebo typů autentizace
- Specification - žadatel si vybírá aktivum, o které má zájem, nebo typ autentizace, který chce použít
- Request - adresát posílá žadateli požadavek na autentizaci
- Response - autentizace žadatele
- Finish - adresát ukončuje transakci a v případě úspěchu posílá požadované aktivum

1.3.2 Formát AVP

Formát AVP je vidět na obr. 1.8.

Pole *Type* určuje typ AVP. Tyto typy se dělí do 3 kategorií:

a) pole *Type* v rozsahu 0-127 - krátké AVP (short - SAVP). Tato AVP jsou určena pro přenos krátkých bloků dat, například adres. Pole *Length* zabírá 1 byte, proto maximální délka pole *Value* je 255 bytů.



Obr. 1.8: Formát AVP

b) pole *Type* v rozsahu 128-191 - dlouhé AVP (long - LAVP). Tato AVP jsou určena pro přenos dlouhých bloků dat, například aktiv. Pole *Length* zabírá 2 byty, proto maximální délka pole Value je 65535 bytů.
c) pole *Type* v rozsahu 192-255 - kontejnerové AVP (container - CAVP). Tato obsahují několik jiných AVP libovolného typu (SAVP, LAVP i vnořené CAVP). Pole *Length* zabírá 2 byty, proto maximální celková délka vnořených AVP je 65535 bytů.

1.3.3 Přehled AVP

Dále je uveden stručný přehled typů AVP. Podrobnější informace lze nalézt v [3].

Jména entit

Jména entit popisují AVP v tabulce 1.1. Tato jména mohou být například ve formátu e-mailové adresy.

Tab. 1.1: AVP se jmény entit

Název AVP	Význam AVP
NAME_SUP_G	Globální jméno žadatele
NAME_PRO_G	Globální jméno poskytovatele
NAME_AUT_G	Globální jméno autentizátoru
NAME_ACC_G	Globální jméno účtovatele
NAME_SUP_L	Lokální jméno žadatele
NAME_PRO_L	Lokální jméno poskytovatele
NAME_AUT_L	Lokální jméno autentizátoru
NAME_ACC_L	Lokální jméno účtovatele

Adresy zařízení

Adresy zařízení popisují AVP v tabulce 1.2. Adresa může představovat adresu IPv4 délkou 4 byty, MAC adresu s délkou 6 bytů, adresu IPv6 s délkou 16 bytů nebo číslo portu s délkou 2 byty.

Tab. 1.2: AVP s adresami zařízení

Název AVP	Význam AVP
ADDR_SUP_G	Globální adresa žadatele
ADDR_PRO_G	Globální adresa poskytovatele
ADDR_AUT_G	Globální adresa autentizátoru
ADDR_ACC_G	Globální adresa účtovatele
ADDR_SUP_L	Lokální adresa žadatele
ADDR_PRO_L	Lokální adresa poskytovatele
ADDR_AUT_L	Lokální adresa autentizátoru
ADDR_ACC_L	Lokální adresa účtovatele

Kódy metod

Autentizační metody popisují AVP EAP a LAM, viz tab. 1.3.

Tab. 1.3: AVP s kódy metod

Název AVP	Význam AVP
EAP	Autentizační metoda EAP
LAM	Lokální autentizační metoda

AVP EAP obsahuje typ EAP autentizace. Tyto typy jsou popsány v [6]. AVP LAM obsahuje lokální typ autentizace. Jednotlivé typy jsou přiděleny lokální autoritou a nejsou blíže specifikovány.

Pro účely autentizace jsou vyhrazeny AVP v rozsazích 96-127 (short AVP), 176-191 (long AVP) a 240-255 (container AVP). Lokální autorita je může využívat pro implementaci vlastních autentizačních metod.

Varianty protokolu

Globální nebo lokální variantu protokolu umožňují svjednat AVP typu GVP a LVP, viz tab. 1.4.

Globální varianta je identifikována šestimístným číslem ve formátu XXXXYY, kde první čtyři číslice XXXX představují číslo RFC, ve kterém je varianta definována, a poslední dvě číslice YY představují pořadí varianty v tomto RFC počínaje číslem 1. Defaultní varianta ACP-VSA má kód 000000. Kódy lokálních variant protokolu určuje administrátor. Kódy jsou v AVP uloženy jako textové řetězce.

Tab. 1.4: AVP popisující variantu protokolu

Název AVP	Význam AVP
GVP	Globální varianta protokolu
LVP	Lokální varianta protokolu

Kódy aktiv

Kódy aktiv popisují AVP v tabulce 1.5.

Tab. 1.5: AVP s kódy aktiv

Název AVP	Význam AVP
ASSET_G	Globální kód aktiva
ASSET_L	Lokální kód aktiva

Globální kód aktiva učuje AVP ASSET_G. Obsahuje jeden oktet, který může nabývat hodnot 0=implicitní aktivum nebo 1=autentizace entity. Lokální aktiva jsou členěna do stromu, přičemž AVP typu ASSET_L určuje cestu tímto stromem. Každý uzel stromu obsahuje nejvýše 256 větví identifikovatelných hodnotou jednoho oktetu. Cesta stromem je tak určena posloupností oktetů. Pokud žadatel nezná cestu k požadovanému aktivu, tak se pomocí zprávy Offer od poskytovatele dozví aktiva v aktuálním uzlu stromu. Zprávou Specification si pak jedno z nich vybere. Několikanásobnou iterací Offer/Specification tak může specifokovat požadované aktivum.

Výstupy transakce

Výsledek běhu transakce je poskytovatelem odeslán ve zpráve Finish. V tabulce 1.6 jsou uvedena AVP, která obsahují výsledek transakce.

Tab. 1.6: AVP obsahující výstup transakce

Název AVP	Význam AVP
RESULT	Výsledek transakce
PROVE	Dokazovací faktor žadatele
VERIF	Verifikační faktor žadatele
FORM_START	Obsah zprávy Start

AVP RESULT obsahuje informaci o úspěchu nebo neúspěchu transakce obsažené v jednom oktetu. Hodnota 0 znamená povolení přístupu, hodnota 2 znamená

zamítnutí přístupu a hodnota 1 vyjadřuje skutečnost, že transakce skončila, ale autentizace ještě neproběhla.

Pokud požadované aktivum žadateli poskytne jiný portál ACP, tak jsou ve zprávě Finish údaje nutné pro přístup k tomuto portálu. Jsou to adresy zařízení (definované výše), dokazovací faktor žadatele (AVP PROVE), verifikační faktor žadatele (AVP VERIF) případně celý obsah zprávy Start (AVP FORM_START).

Interoperabilita

Pro zajištění interoperability s ostatními autentizačními protokoly jsou definovány AVP obsahující buď celé zprávy nebo AVP z těchto protokolů. Tyto AVP jsou zapsány v tabulce 1.7.

Tab. 1.7: AVP pro zajištění interoperability

Název AVP	Význam AVP
MESS_RADIUS	Zpráva protokolu Radius
MESS_DIAMETR	Zpráva protokolu Diameter
MESS_KERBER	Zpráva protokolu Kerberos
AVP_RADIUS	AVP protokolu Radius
AVP_DIAMETR	AVP protokolu Diameter

Kryptografická primitiva

Přenášené zprávy mohou být zabezpečeny pomocí kryptografických primitiv z povinné sady protokolu TLS. Tato primitiva jsou přenášena uvnitř AVP v tabulce 1.8.

Doplňková AVP

AVP mohou obsahovat popis svého obsahu. Ten může být například zobrazen uživateli na obrazovce. K tomuto účelu slouží AVP v tabulce 1.9.

1.3.4 Mechanismus transakcí

Každá zpráva protokolu ACP přísluší k některé transakci. Transakce je uzavřena mezi ACP serverem žadatele a adresáta a může být tranzitována přes několik tranzitních serverů. Pole *Identifier* určuje transakci, do které zpráva patří. Pole je unikátní v rámci jednoho spojení a směru transakce. Více je vidět na obr. 1.9.

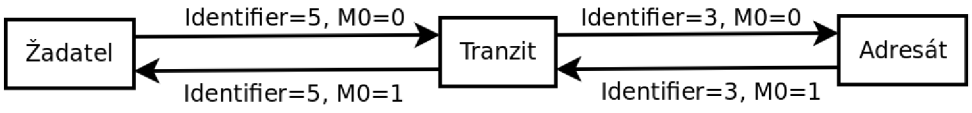
Žadatel vytvořil transakci a přidělil ji hodnotu *Identifier* 5. Transitní server tuto transakci směroval na adresáta a přidělil ji hodnotu *Identifier* 3. Při průchodu zpráv

Tab. 1.8: AVP s kryptografickými primitivy

Název AVP	Význam AVP
INIT	Inicializační vektor
PMS	Premaster secret
CERT	Certifikát
AES	Kryptogram zašifrovaný šifrou AES
ENC	Kryptogram s inicializačním vektorem
HMAC	Autentizační kód zprávy
MAC	Data s autentizačním kódem
RSA	Kryptogram zašifrovaný šifrou RSA
PSS	Digitální podpis
SIG	Data s digitálním podpisem
CRYPT	Kryptografický kontejner

Tab. 1.9: Doplňková AVP

Název AVP	Význam AVP
TXT	Popis předchozího AVP
EAP_TX	AVP EAP s připojeným popisem
LAM_TX	AVP LAM s připojeným popisem
ASSET_G_TX	AVP ASSET_G s připojeným popisem
ASSET_L_TX	AVP ASSET_L s připojeným popisem
LIST	seznam SAVP stejného typu a délky



Obr. 1.9: Mechanismus transakcí

přes tranzitní server je tedy nutné změnit hodnotu pole *Identifier*. Pokud by byla vytvořena jiná transakce opačným směrem (žadatel a adresát by se prohodily), tak pro ni může být přidělena stejná hodnota pole *Identifier*. Transakce jsou pak odlišitelné hodnotou bitu *M0*, který určuje, zda je zpráva odeslána žadatelem nebo adresátem. Díky tomu si každý portál ACP přiděluje hodnoty *Identifier* sám a nemusí se domlouvat s ostatními portály ACP.

Transakci zahajuje žadatel zprávou Start a ukončuje adresát zprávou Finish. Portál při obdržení zprávy Start určí, zda je adresátem transakce a pokud ne, tak

kam má transakci směrovat. Následně pro toto odchozí spojení určí dosud nepřidělenou hodnotu pole *Identifier*. Při obdržení zprávy Finish vymaže portál záznamy o transakci. Navíc má portál nastaven časový limit pro transakci po němž dojde k jejímu zrušení.

1.4 Testování protokolu ACP

1.4.1 Testované vlastnosti protokolu

Abych mohl navrhnout testovací software, je nutné vědět, jaké vlastnosti protokolu budu testovat. Především je nutné otestovat navazování transakcí, předávání zpráv v rámci navázané transakce a její ukončení. Je nutné testovat jak regulérní běh protokolu, tak mimořádné situace jako ztráta nebo zdvojení zprávy, případně předčasné ukončení transakce časovačem. Také je vhodné otestovat možný průnik útočníka do transakce navázané jiným žadatelem.

Dále je nutné testovat směrování zpráv, a to i přes více portálů. Každý portál se musí být schopný rozhodnout, kam bude zprávy směrovat a jak přepsat identifikátor transakce při předávání zprávy. Opět je nutné testovat i různé mezní stavy, jako například předčasné ukončení transakce.

Pokud bude mechanismus předávání zpráv funkční, tak je vhodné testovat různé možné varianty protokolu ACP. Protokol je otevřený a umožňuje různé informace předávat v různých zprávách. Je například možné přímo ve zprávě Start specifikovat požadované aktivum, nebo je možné nechat si od serveru zaslat seznam aktiv. Některá aktiva mohou být poskytnuta bez nutnosti autentizace, jiná aktiva autentizaci vyžadují a může být vyžadována různá autentizace dle druhu aktiv. Proto navrhuji následující scénáře běhu protokolu:

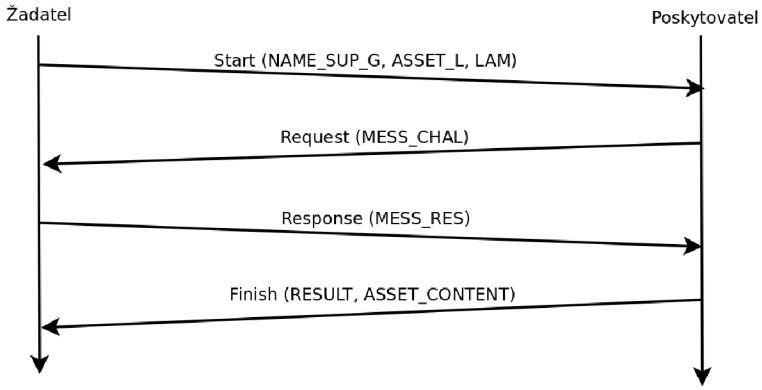
1.4.2 První testovací scénář

První scénář protokolu je vidět na obr. 1.10. Jedná se o nejjednodušší možný běh protokolu, jak je popsán v [3] v sekci ACP-VSA. Účelem tohoto testu je otestovat základní funkčnost protokolu ACP, funčnost navázání a zrušení transakce a předávání zpráv. Scénář bude provádět autentizaci stálým heslem.

Žadatel ve zprávě Start uvede v AVP NAME_SUP_G jméno žadatele, v AVP ASSET_L identifikátor požadovaného aktiva a v AVP LAM kód autentizační metody. Poskytovatel ve zprávě Request odešle autentizační výzvu (náhodnou posloupnost oktetů v AVP MESS_CHAL). Žadatel na ni odpoví ve zprávě Response hodnotou SHA256 (heslo | výzva) v AVP MESS_RES.

Pokud autentizace proběhne správně, typ autentizace bude pro požadované aktivum dostačovat a žadatel bude mít přístupová práva k požadovanému aktivu, tak poskytovatel ve zprávě Finish aktivum odešle v AVP ASSET_CONTENT.

Pro účely testování jsem zavedl AVP MESS_CHAL s hodnotou typu 178, AVP MESS_RES s hodnotou typu 179 a ASSET_CONTENT s hodnotou typu 176. Všechny AVP tak leží v rozsahu typů rezervovaných pro lokální autentizační metody.



Obr. 1.10: Scénář 1

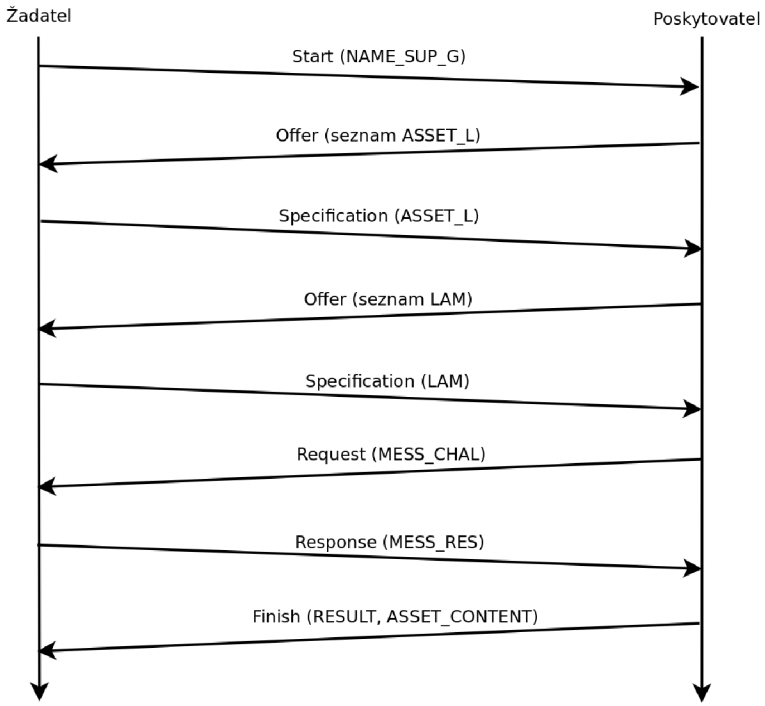
1.4.3 Druhý testovací scénář

Druhý scénář je vidět na obr. 1.11. Oproti prvnímu scénáři obsahuje navíc sjednání požadovaného aktiva a autentizační metody. Smyslem tohoto testu je otestovat právě toto sjednávání. Vlastní autentizace probíhá stejně jako v prvním testovacím scénáři.

Žadatel ve zprávě Start uvede jen své jméno v AVP NAME_SUP_G. Poskytovatel mu ve zprávě Offer odpoví seznamem dostupných aktiv zapsaným jako několik AVP ASSET_L. Žadatel si jedno z nich vybere a jeho identifikátor odešle ve zprávě Specification v AVP ASSET_L. Následně poskytovatel určí seznam přípustných autentizací pro toto aktivum a pošle ho žadateli ve zprávě Offer jako seznam AVP LAM. Žadatel si jednu z nich vybere a ve zprávě Specification odešle její identifikátor v AVP LAM. Další průběh (autentizace a odeslání požadovaného aktiva) je shodný s prvním testovacím scénářem.

1.4.4 Třetí testovací scénář

Třetí scénář je vidět na obr. 1.12. V tomto testovacím scénáři se testuje zřetězení transakcí ve spojení s autentizací asymetrickou šifrou. Žadatel si nejdříve vygeneruje dočasnou dvojici RSA klíčů [*VKtmp*, *SKtmp*]. Veřejný klíč *VKtmp* pošle autentizačnímu serveru v AVP PUB_KEY_SUP ve zprávě Start spolu se svým jménem v AVP NAME_SUP_G, kódem požadovaného aktiva (tj. certifikátu) v AVP



Obr. 1.11: Scénář 2

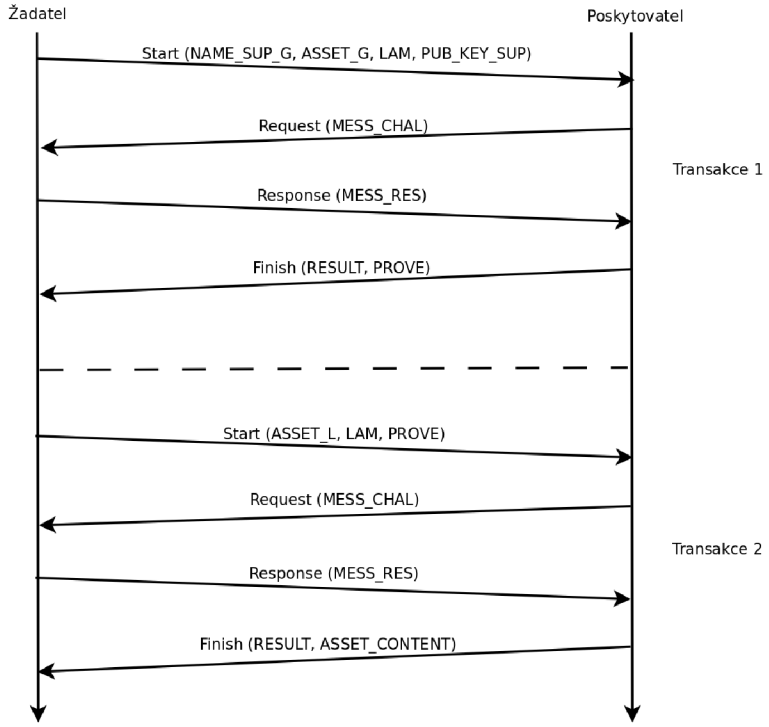
ASSET_G a typem autentizace v AVP LAM. Pak proběhne autentizace žadatele pomocí zpráv Request a Response. Ve zprávě Request odešle autentizační server autentizační výzvu *Vaut* v AVP MESS_CHAL. Žadatel odešle ve zprávě Response v AVP MESS_RES digitální podpis zprávy [*Vaut*, *VKtmp*] podepsaný soukromým klíčem žadatele *SKž*. Autentizační server ověří platnost tohoto podpisu (má k dispozici veřejný klíč žadatele *VKž*). Případný útočník může modifikovat zprávu předávanou autentizačnímu serveru, ponechat autentizační údaje, ale podvrhnout vlastní dočasný veřejný klíč. který by pak autentizační server podepsal a umožnil tak útočníkovy přístup do ACP sítě jménem žadatele. Proto se v autentizaci podepisuje i klíč *VKtmp*, aby se tomuto zabránilo.

Autentizační server vydá certifikát [*NAME_SUP_G*, *VKtmp*, *INFO*] podepsaný svým vlastním soukromým klíčem *SKaut* a ve zprávě Finish ho pošle žadateli v AVP PROVE. Pole *INFO* obsahuje dodatečné informace od autentizačního serveru, například čas expirace certifikátu a přístupová práva žadatele.

Následně žadatel naváže spojení se serverem poskytujícím požadované aktivum, ve správě Start odešle certifikát získaný od autentizačního serveru v AVP PROVE, identifikátor požadovaného aktiva v AVP ASSET_L a typ autentizace v AVP LAM. Server ověří, zda je certifikát platný (má k dispozici veřejný klíč autentizačního serveru *VKaut*), zda má žadatel přístupová práva k požadovanému aktivu a vygeneruje výzvu *Vs*, kterou pošle žadateli ve zprávě Request v AVP MESS_CHAL. Žadatel

výzvu podepíše svým dočasným soukromým klíčem *VKtmp* a tento podpis odešle ve zprávě Response v AVP MESS_RES. Server ověří, zda je podpis platný (veřejný klíč *VKtmp* získal z certifikátu) a pokud ano, tak ve zprávě Finish odešle požadované aktivum.

Zavedl jsem AVP PUB_KEY_SUP s hodnotou 177 obsahující veřejný klíč žadatele. AVP spadá do rozsahu rezervovaných typů, jak je popsáno v [3].



Obr. 1.12: Scénář 3

Testovací ACP server umožní pro jednoduchost připojení jen k jednomu nadřízenému ACP serveru. Proto se v tomto testu dopustím zjednodušení - autentizační server a server poskytující aktiva budou totožné. Žadatel tedy naváže dvě transakce se stejným ACP serverem, ten si však mezi transakcemi nebude pamatovat žádné informace, bude se tedy vůči žadateli tvářit jako dva nezávislé servery.

1.5 Požadované vlastnosti testovacího softwaru

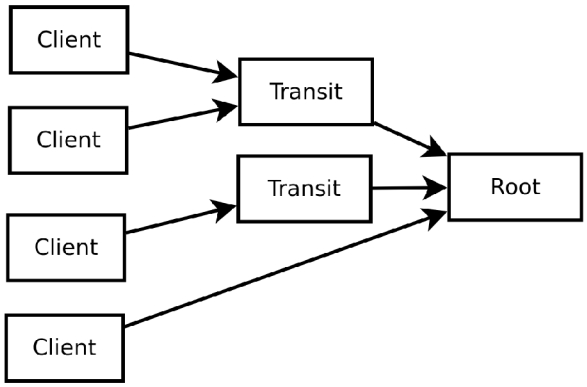
Z výše popsaného lze odvodit, jaké vlastnosti by měl mít testovací software a jaké naopak nejsou důležité.

Především tento testovací software bude používán pro testování a ne pro skutečný provoz. Není tedy nutné dbát na rychlost zpracování zpráv, protože server nebude příliš zatěžován. Není ani nutné zajistit tak vysokou míru bezpečnosti a spolehlivosti serveru, jaká by byla nutná ve skutečném provozu.

Naopak je vhodné, aby byl software snadno přenositelný mezi různými platformami. To umožní provádět testy na rozsáhlejší ACP síti. Dále je vhodné, aby software, a zejména modul zajišťující vyhodnocení přijatých zpráv a reakci na ně, byly co nejsnáze modifikovatelné. V ideálním případě by tento modul měl být umístěn mimo vlastní software a jít tak modifikovat beze změny tohoto softwaru. Půjde tak snadno připravit několik různých scénářů běhu protokolu ACP.

1.5.1 Hierarchie serverů

Protokol ACP umožňuje vytvářet obecné složité struktury mezi sebou komunikujících portálů. Abychom situaci poněkud zjednodušili, navrhli jsme spolu s doc. Ing. Burdou, CSc. hierarchickou síť, jak je vidět na obr. 1.13. Implementaci portálu ACP jsem nazval ACP server. Rozlišuji 3 typy serverů - Root, Transit a Client.



Obr. 1.13: Typy serverů

Typ serveru Root

Root server je nejvýše postavený server v hierarchii. Jedná se o klíčový bod celé sítě, jeho výpadek způsobí výpadek celé sítě. Spravuje aktiva a provádí autentizaci a autorizaci. Je koncovým uzlem při směrování zpráv typu Start, proto je adresátem všech transakcí.

Typ serveru Transit

Transitní server sám není účastníkem žádné transakce, pouze předává zprávy mezi jinými servery. Může proto vykonávat funkci brány, která převádí jeden druh spojení na jiný. Například čtečka u dveří, do které uživatel přes USB připojí svůj autentizační token, bude připojená k síti a pude tak převádět zprávy z USB rozhraní do TCP/IP.

Typ serveru Client

Klient je iniciátorem transakcí. Zde uživatel zadává požadavky na aktiva a autentizační údaje. Příkladem jsou tedy autentizační karty a tokeny obsahující procesor s implementací portálu ACP nebo pracovní stanice přistupující k aktivům na školním či firemním serveru.

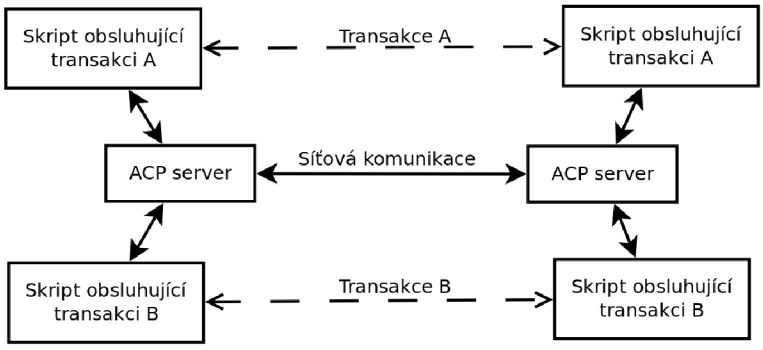
1.6 Struktura programu

ACP server je naprogramován v jazyce Java. Tento jazyk jsem zvolil, protože zajistí snadnou přenositelnost programu mezi různými platformami. Navíc obsahuje bohatou standardní knihovnu, viz [7].

- balíček *javax.swing* umožňuje snadno realizovat grafické uživatelské rozhraní (GUI) programu
- balíček *java.security* obsahuje potřebná kryptografická primitiva nutná pro autentizaci
- balíček *java.net* má vše, co je nutné pro realizaci síťové komunikace
- balíček *javax.net* obsahuje implementaci SSL/TLS vrstvy

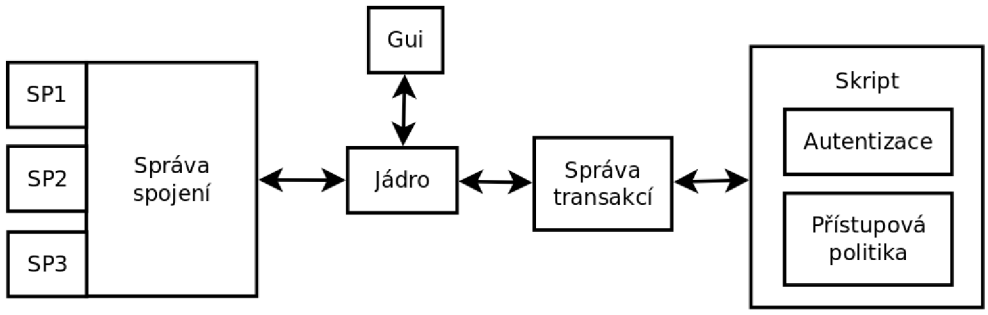
Důležité je, že přenositelný není jen vlastní jazyk Java, ale i celá standardní knihovna s výše uvedenými balíčky. Není proto nutné programovat abstraktní vrstvu nad systémově závislými funkcemi, jak by bylo nutné například v jazyce C++.

Vlastní ACP server řeší jen předávání zpráv po síti a správu transakcí a neobsahuje žádné autentizační metody. Tyto jsou naprogramovány ve skriptovacím jazyce JavaScript a jsou umístěny mimo ACP server. Server spustí skript pro každou transakci a předává mezi nimi zprávy. S autentizačními metodami tak lze snadno experimentovat bez nutnosti měnit vlastní ACP server. Schema předávání zpráv mezi skripty je vidět na obr. 1.14.



Obr. 1.14: Předávání zpráv mezi skripty

Blokové schema programu je vidět na obr. 1.15. ACP server sestává z několika modulů s přesně definovanými vztahy. Postup návrhu struktury programu je popsán například v [8].



Obr. 1.15: Blokové schema ACP serveru

1.6.1 Jádro

Jádro serveru se stará o koordinaci ostatních modulů a není přímo závislé na ostatních modulech, ale jen na definovaném rozhraní těchto modulů. Proto je možné v případě potřeby modul nahradit jiným, případně mezi jádro a modul vložit prostředníka, který zajistí komunikaci mezi modulem a jádrem. Například je možné mezi GUI a jádro vložit dvojici proxy objektů, které si mezi sebou budou předávat zprávy, a vytvořit tak funkci vzdálené správy serveru.

1.6.2 Gui

Grafické uživatelské rozhraní umožňuje uživateli nastavovat parametry serveru a sledovat jeho stav. Konkrétně umožňuje nastavit typ serveru, tedy vybrat mezi kořenovým serverem, tranzitním serverem nebo klientem. Dále umožňuje nastavit typ spojení a adresu nadřízeného serveru, stejně jako rozhraní, na nichž má server nastlouchat příchozím spojením od podřízených serverů.

Grafické uživatelské rozhraní také uživateli zobrazuje seznam navázaných spojení a seznam aktivních transakcí a zobrazuje o nich statistiky. Dále zobrazuje log událostí, které nastaly, včetně přijatých a odeslaných zpráv. Rozhraní je naprogramováno pomocí již zmíněné knihovny Swing.

1.6.3 Správa transakcí

Modul správy transakcí se stará o spravování záznamů o transakcích. Tyto záznamy obsahuji informace nutné pro směrování zpráv a informace o stavu transakcí. Navíc, je li k transakci přidružený spuštěný skript, tak modul zajišťuje jeho správu

a předávání zpráv ACP mezi skriptem a jádrem.

1.6.4 Správa spojení

Modul správy spojení má na starosti doručování zpráv ostatním ACP serverům v síti - jejich kódování a vyslání na straně vysílače a přijetí a dekódování na straně přijímače. Dále modul naslouchá novým spojením od podřízených ACP serverů. Také vytváří a udržuje spojení s nadřízeným serverem.

K modulu správy spojení se připojují moduly jednotlivých druhů spojení SP. Správa spojení je proto nezávislá na použité přenosové technologii, vytvořením nového modulu spojení lze server rozšířit o nový druh přenosového kanálu. V současnosti jsou implementovány dva moduly spojení - klasické TCP proudu a SSL/TLS vrstva nad TCP proudem. Klasický TCP proud je vhodný pro testování, protože umožňuje zachytávat a analyzovat přenášená data. SSL/TLS spoj je naopak vhodný pro skutečné nasazení ACP serveru, obzvláště pokud jsou data přenášena přes veřejnou síť. Zajišťuje totiž bezpečnost přenášených dat. Oba druhy spojení využívají komunikace pomocí socketů, které poskytují abstrakci nad různými přenosovými technologiemi. Více o socketech se lze dočíst v [9].

1.6.5 Skript

Ve skriptu jsou naprogramovány scénáře protokolu ACP. Obsahuje modul autentizace a přístupové politiky. Skript na straně kořenového serveru provádí správu aktiv, autentizaci žadatele i autorizaci přístupu k aktivům. Skript na straně klienta simuluje chování žadatele. Posílá kořenovému serveru požadavky na aktiva a autentizační údaje žadatele.

Skript pracuje jako event handler. Po zahájení transakce spuštěním příkazu žadatele jeho skript odešle zprávu Start poskytovateli a čeká, dokud jej ACP server neinformuje o příchodu odpovědi. Na ni opět odpoví zprávou a bude čekat na odpověď. Tento cyklus se neustále opakuje do ukončení transakce. Podobně ACP server poskytovatele po obdržení zprávy Start vytvoří novou instanci skriptovacího enginu, spustí v něm skript a předá mu zprávu. Skript zprávu zpracuje, odpoví žadateli a bude čekat na jeho odpověď.

Skript je snadno modifikovatelný, proto lze libovolně měnit naprogramované scénáře, autentizační metody i přístupové politiky. Díky tomu je server modulární, výměnou skriptu lze kompletně změnit jeho chování a experimentovat s novými druhy autentizace a přístupových politik.

Skripty jsou naprogramovány v jazyce JavaScript a interpretovány enginem Rhino. Tento engine je vestavěný v JRE 1.6 a vyšším, není tedy nutné k ACP serveru při-

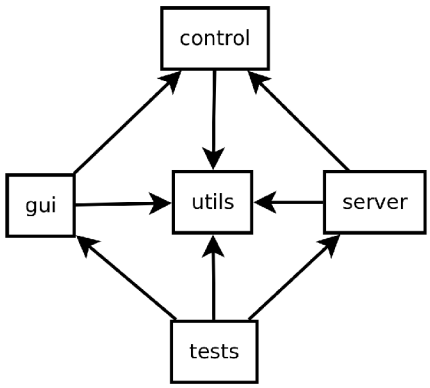
dávat dodatečné knihovny. Rhino umožňuje skriptům využívat standardní knihovnu jazyka Java. Ve skriptech se využívá především kryptografický balíček *java.security* pro implementaci autentizačních metod. Podrobnosti o enginu Rhino lze nalézt v [10].

1.7 Technická dokumentace

V diplomové práci jsou popsány jen základní třídy, jejich vazby a myšlenky použité v návrhu programu. Zdrojový kód je bohatě (anglicky) komentovaný a pomocí nástroje Javadoc je z něj vygenerována technická dokumentace uložená na přiloženém CD.

1.7.1 Rozdělení programu do balíčků

ACP server je rozdělen do několika balíčlů. Každý z nich obsahuje jeden nebo více modulů uvedených v blokovém schematu na obr. 1.15. Závislosti mezi balíčky jsou pak naznačeny na obr. 1.16.

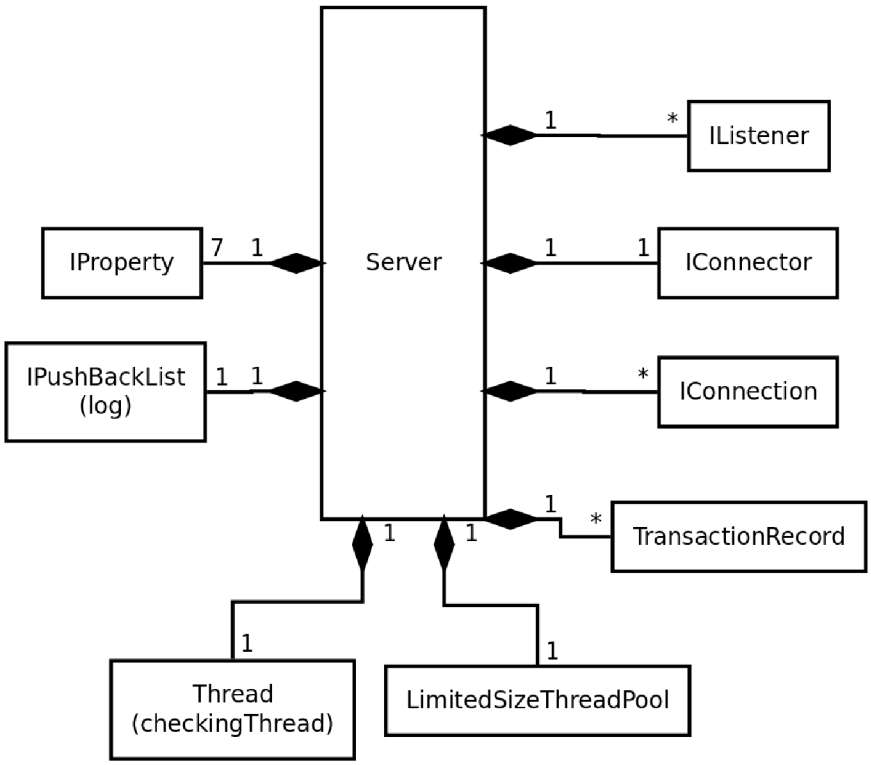


Obr. 1.16: Závislosti balíčků

Balíček server

Tento balíček zajišťuje výkonnou část ACP serveru. Jeho struktura je vidět na obr. 1.17. Obsahuje jádro, moduly správy spojení a správy transakcí.

Hlavní úlohu hraje třída *Server* popsaná v sekci 1.7.2. Tato třída přijímá pokyny od GUI. Pomocí svých vlastností prezentuje stav, v němž se nachází, a aktuální nastavení serveru. Vlastnosti jsou třídy implementující rozhraní *IProperty*. Dále třída vede log o událostech, které nastaly, v seznamu implementujícím rozhraní *IPushBackList*. Třída také vlastní posluchače, kteří naslouchají příchozím spojením od podřízených ACP serverů. Tito posluchači implementují rozhraní *IListener* a jsou blíže



Obr. 1.17: Schéma balíčku server

popsáni v sekci 1.7.6. Dále třída *Server* vlastní jednoho spojovatele zajišťujícího spojení na nadřazený ACP server. Tento spojovatel implementuje rozhraní *IConnector* a je blíže popsán v sekci 1.7.7. Posluchači a spojovatelé vytvářejí spojení, které předávají třídě *Server* k obsluze. Spojení mají na starosti přijímání a odesílání zpráv okolním ACP serverům. Spojení implementují rozhraní *IConnection* a jsou blíže popsána v sekci 1.7.5.

Třída *Server* si vede záznamy o transakcích, které zprostředkovává. Tyto záznamy reprezentují instance tříd *TransactionRecord* a jsou uloženy v tabulce *transactionRecordMap*. Třída *Server* pomocí nich směruje nabo zpracovává zprávy, které obdrží od spojení.

Zpracování zpráv a některou obslužnou činnost provádí třída asynchronně. K tomu jí slouží vlákno *checkingThread*, které periodicky kontroluje stav spojení a transakcí a odstraňuje je, pokud již nejsou potřeba. Třída *LimitedSizeThreadPool* zajišťuje zpracování požadavků (zpráv a příkazů). Blíže je popsána v sekci 1.7.4.

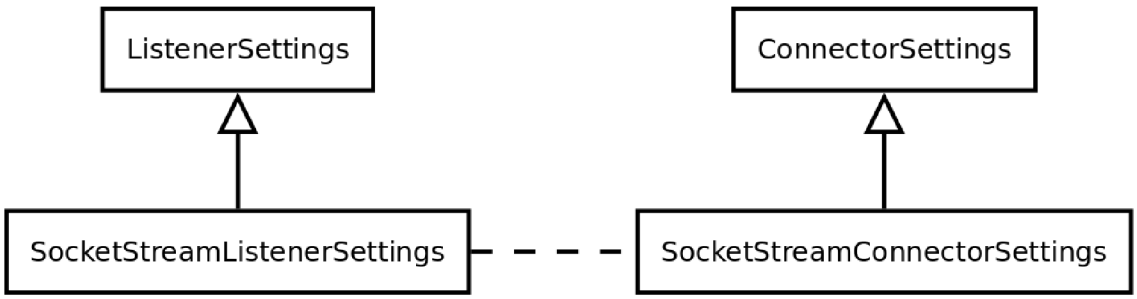
Balíček gui

Tento balíček obsahuje kód zodpovědný za tvorbu uživatelského rozhraní. Přes rozhraní definované v balíčku *control* předává příkazy ACP serveru a zobrazuje jeho stav. Grafické uživatelské rozhraní je naprogramováno pomoví knihovny Swing.

Balíček control

Tento balíček obsahuje rozhraní mezi balíčkem *gui* a balíčkem *server*. Rozhraní *IServerControl* slouží pro komunikaci mezi serverem a GUI. Obsahuje metody pro ovládání serveru a tzv. vlastnosti, které poskytují informace o stavu serveru. Vlastnosti implementují rozhraní *IProperty*. Toto rozhraní jednak umožňuje zjistit aktuální hodnotu vlastnosti a také umožňuje zaregistrovat posluchače vlastnosti implementujícího rozhraní *IPropertyListener* který je informován o jejích změnách. Podobně rozhraní *IPushBackList* umožňuje přistupovat k seznamu hodnot. Toto rozhraní se používá pro realizaci logu. Balíček navíc obsahuje deklaraci tříd s nastavením serveru. Třídy s nastavením implementují rozhraní *IXmlPersistence* a mohou tak být uloženy do XML souboru a opět načteny. Nastavení tak může být permanentně uloženo do konfiguračního souboru při ukončování ACP serveru a opět načteno při jeho opětovném spuštění.

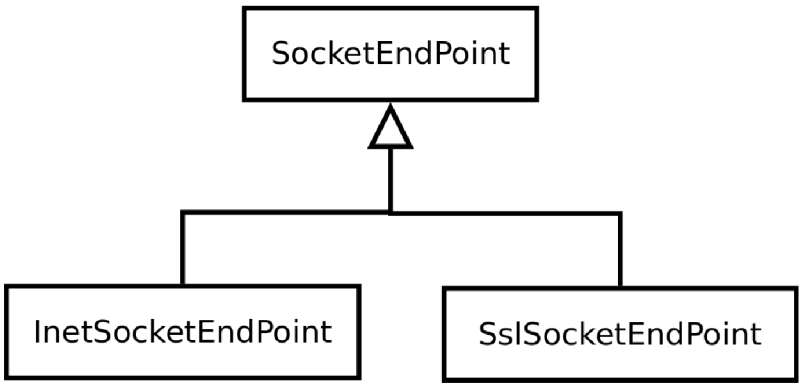
Konfigurace serveru je řešena tak, že každé konfiguovatelné třídě v balíčku *server* odpovídá jedna třída s nastavením v balíčlu *control*. Například rozhraní *IConnector* čte konfiguraci z obecné třídy *ConnectorSettings*. Jeho potomek *SocketStreamConnector* již vyžaduje instanci třídy *SocketStreamConnectorSettings*. Podobně rozhraní *IListener* čte konfiguraci z obecné třídy *ListenerSettings*. Jeho potomek *SocketStreamListener* vyžaduje instanci třídy *SocketStreamListenerSettings*. Třídy s nastavením jsou vidět na obr. 1.18.



Obr. 1.18: Nastavení spojovatelů a posluchačů

Typ spojovatele, posluchače a spojení popisuje enumerace *ConnectionType*. Třída *ConnectorSettingFactory* má na starosti vytváření spojovatele podle typu spojení a třída *ListenerSettingFactory* vytváří posluchače podle typu spojení.

Adresu, na kterou se spojovatel snaží připojit nebo na které posluchač naslouchá, určují koncové body - potomci třídy *SocketEndPoint*. V současnosti existují dva potomci - *InetSocketEndPoint* představující koncový bod TCP proudu a *SslSocketEndPoint* představující koncový bod SSL/TLS spoje. Hierarchie koncových bodů je vidět na obr. 1.19.

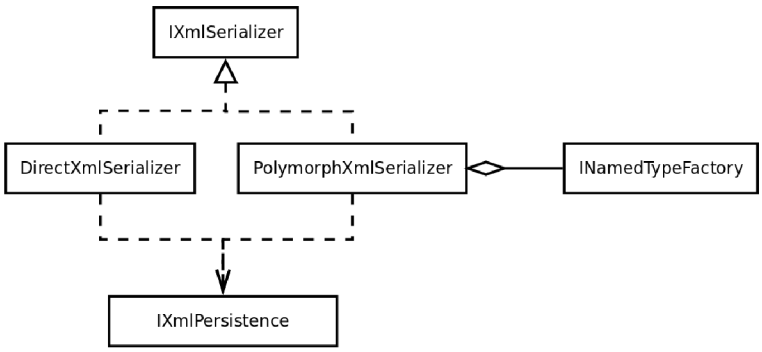


Obr. 1.19: Koncové body

Balíček utils

Tento balíček obsahuje pomocné třídy používané ostatními balíčky. Jsou zde obsaženy třídy, které nelze vložit do jiných balíčků.

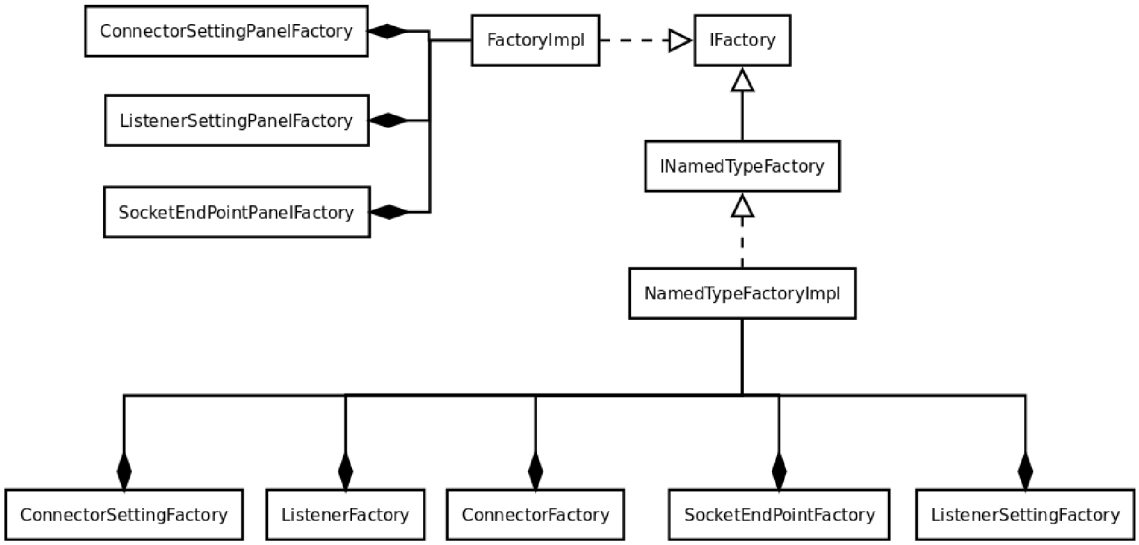
V tomto balíčku jsou mimo jiné třídy zajišťující serializaci objektů, jak je vidět na obr. 1.20. Rozhraní *IXmlPersistence* implementují všechny třídy, které je možno uložit do XML elementu. Podobně rozhraní *IXmlSerializer* představuje XML serializátor, který umožňuje uložit zadaný objekt do XML elementu a vytvořit nový objekt načtený z XML elementu. Třída *DirectXmlSerializer* implementuje rozhrazí *IXmlSerializer* a umožňuje tak vytvářet instance tříd implementujících rozhraní *IXmlPersistence*. Při načítání objektu z XML vytváří vždy stejnou, předem danou, třídu. Naproti tomu třída *PolymorphXmlSerializer* spolu se serializovanými daty ukládá identifikátor uložené třídy. Při načítání tak lze vytvořit instanci třídy, která byla uložena, z více možných. K vytváření tříd využívá továrnu na objekty *INamedTypeFactory*.



Obr. 1.20: XML serializace

Továrna na objekty slouží pro vytváření objektů podle jejich identifikátoru. Rozhraní *IFactory* umí právě toto - vytvořit novou instanci třídy s daným identifiká-

torem a vrátit seznam všech známých identifikátorů. Rozhraní *INamedTypeFactory* navíc umí vrátit identifikátor typu podle jeho názvu. Tato rozhraní jsou implementovány třídami *FactoryImpl* a *NamedTypeFactoryImpl*. Třídy *ConnectorSettingPanelFactory*, *ListenerSettingPanelFactory*, *SocketEndpointPanelFactory*, *ConnectorSettingFactory*, *ListenerFactory*, *ConnectorFactory*, *SocketEndPointFactory* a *ListenerSettingFactory* ve svých statických proměnných drží instance továren vytvářejících objekty odpovídající názvům těchto tříd. Diagram tříd je vidět na obr. 1.21.



Obr. 1.21: Továrny na objekty

Balíček tests

Zde jsou obsaženy jednotkové a integrační testy programu. Testy umožňují prověřit funkčnost jednotlivých komponent programu a při jejich modifikaci snadno lokalizovat případné chyby.

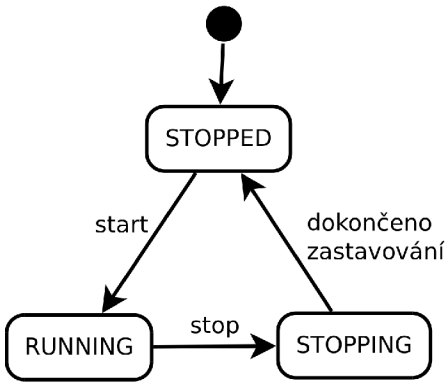
Třídy *AvpTest*, *MessageTest*, *SocketStreamConnectionTest* a *StreamPackagerTest* obsahují jednotlivé testy spustitelné v prostředí JUnit. Třída *AllTests* je přímo spustitelná a postupně zavolá všechny výše uvedené testy.

1.7.2 Třída Server

Základní třídou ACP serveru je *acp.server.Server*. Tato třída je zodpovědná za vlastní řízení serveru, registraci transakcí a předávání zpráv mezi spojeními, případně mezi spojením a interpreterem skriptu. Třída *Server* implementuje rozhraní *IServerControl*, skrze které přijímá příkazy od GUI. Rozhraní a třída definují vlastnosti *serverState*, *serverType*, *transactionInfoCollection*, *connectionInfoCollection*, *parentConnectorInfo*, *listenerInfoCollection* a *scriptPath*. Každá vlastnost publikuje

určitou informaci a umožňuje zaregistrovat jednoho nebo více posluchačů, kteří jsou informováni o její změně. Tím je dosaženo toho, že GUI není závislé na serveru a server není závislý na GUI. Je tak možné vytvořit jinou implementaci GUI nebo serveru, případně mezi server a GUI vložit mezivrstvu, která např. zajistí předávání příkazů a informací přes síť.

Server se může nacházet v jednom ze 3 stavů, jak je vidět na obr. 1.22:



Obr. 1.22: Stavový diagram serveru

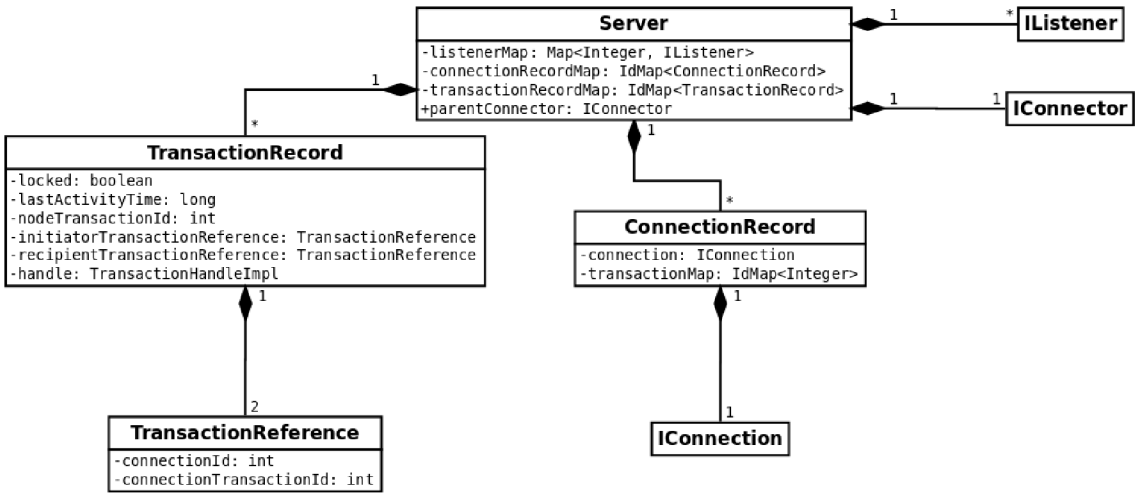
Výchozí stav je *STOPPED*. Pouze v tomto stavu je možné měnit nastavení serveru. Po zavolání metody *start* server podle aktuálního nastavení aktivuje své dílčí komponenty a přejde do stavu *RUNNING*. V tomto stavu provádí svou činnost. Zavoláním metody *stop* se aktivuje proces zastavování serveru. Protože během něho je nutné čekat na ukončení jednotlivých komponent serveru a tyto potřebují volat jeho metody, tak monitor serveru nemůže být během čekání zamčený. Proto server přejde do stavu *STOPPING*, kdy je schopen obsluhovat tyto komponenty a stále mu nelze z vnějšku měnit nastavení. Po ukončení všech dílčích komponent přejde do výchozího stavu *STOPPED*.

1.7.3 Typy serverů

Každý server má nastaven jeden ze 3 typů vyjmenovaných ve výčtu *ServerType*. Jsou to *CLIENT*, *TRANSIT* a *ROOT*. Jejich vzájemné vztahy jsou zachyceny na obr. 1.13.

1.7.4 Prostředky serveru

Třída *Server* spravuje prostředky ACP serveru. Mezi tyto prostředky patří posluchači, spojovatelé, navázaná spojení a transakce, jak je vidět na obr. 1.23.

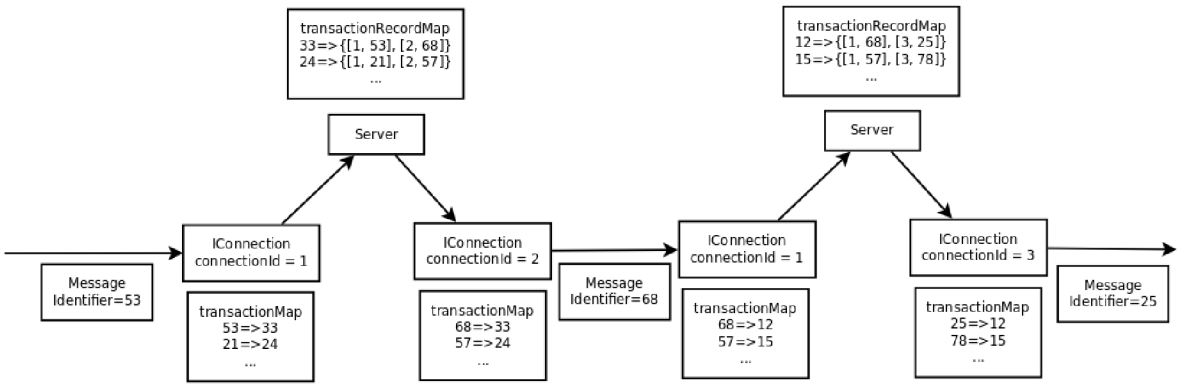


Obr. 1.23: Prostředky serveru

Posluchač naslouchá příchozím spojením od jiných (podřízených) ACP serverů. Jakmile dojde k navázání spojení, tak vytvoří objekt implementující rozhraní *IConnection* a přidá jej do mapy spojení v ACP serveru. Spojovatel se snaží navázat spojení s nadřízeným ACP serverem. Jakmile se mu to podaří, tak podobně jako posluchač vytvoří objekt spojení a přidá jej do mapy spojení ACP serveru. Jakmile se tedy podaří navázat spojení, tak budou existovat dva objekty spojení - jeden v nadřízeném ACP serveru vytvořený posluchačem a jeden v podřízeném ACP serveru vytvořený spojovatelem. Tyto objekty zajistí předávání zpráv mezi oběma ACP servery.

Každá zpráva protokolu ACP je obsažena v určité transakci, která je identifikována polem *Identifier*. Podle tohoto pole ACP server se zprávami nakládá (směruje je). U každého spojení je udržována mapa transakcí (*transactionMap*), která dané hodnotě *Identifier* přiřazuje jedinečné ID transakce v rámci ACP serveru. Server si vede mapu transakcí (*transactionRecordMap*), která mimo jiné v každém záznamu obsahuje dvě dvojice [ID spojení, ID transakce v rámci spojení] - jednu dvojici pro iniciátora a jednu pro adresáta transakce. Pomocí nich lze dohledat, kam má být zpráva směrována.

Princip předávání zpráv je vidět na obr. 1.24. Pokud zpráva není určena pro daný server (tj. pokud je server tranzitní), tak server dle této mapy vyhledá spojení k adresátovy a přepíše ve zprávě hodnotu pole *Identifier*. Zpráva pak putuje k dalšímu ACP serveru, který ji buď zpracuje, pokud je adresátem, nebo přepošle dál. Záznamy v mapách se vytváří při obdržení zprávy Start a ruší při obdržení zprávy Finish, případně po překročení stanovené maximální doby tvrání transakce.



Obr. 1.24: Tok zpráv

Třída LimitedSizeThreadPool

Thread pool je návrhový vzor zajišťující zpracování úloh hromadících se ve frontě jedním nebo více předvytvořenými vlákny. Ty postupně odebírají úlohy z fronty a vykonávají je, úlohy jsou tak zpracovávány sériovo-paralelně. Optimální je mít počet vláken shodný s počtem procesorů nebo jader v počítači. Aby úlohy mohly být zpracovány paralelně, tak musejí být na sobě nezávislé a co nejméně přistupovat ke sdíleným prostředkům.

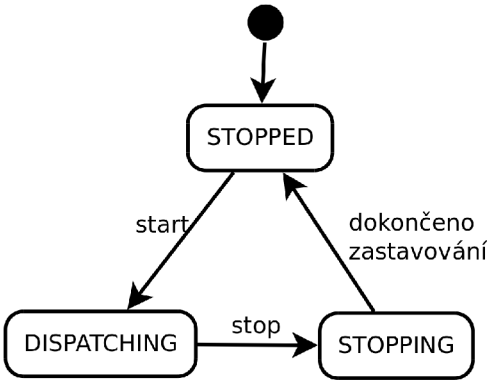
Třída *LimitedSizeThreadPool* je implementace výše uvedeného návrhového vzoru. Zpracovávané úlohy se dělí na dvě skupiny - normální a důležité. Třída má definovanou maximální velikost fronty. Pokud je fronta zaplněná, tak do ní nelze vložit další normální úlohu. Na důležité úlohy se toto omezení nevztahuje a lze je vložit vždy, i když dojde k překročení maximální velikosti fronty. Normální úlohy například zpracovávají příchozí zprávy, důležitá úloha je například volání spojovatele pro spojení s nadřízeným ACP serverem.

1.7.5 Spojení

Každé spojení implementuje rozhraní *IConnection*. Spojení je zodpovědné za doručování zpráv mezi dvěma ACP servery a vytváří potřebnou abstrakci pro třídu *Server*, která tak není závislá na typu spojení. Jedno fyzické spojení (např. TCP proud) je udržováno mezi dvěma instancemi stejné třídy implementující rozhraní *IConnection*, přičemž každý je na jedné straně spojení, tedy na různých ACP serverech. Spojení obsahuje frontu zpráv na odeslání a registrovaný handler, který je informován o událostech, které nastaly. Je-li spojení vytvořeno ve třídě *Server*, pak je handlerem právě tato třída.

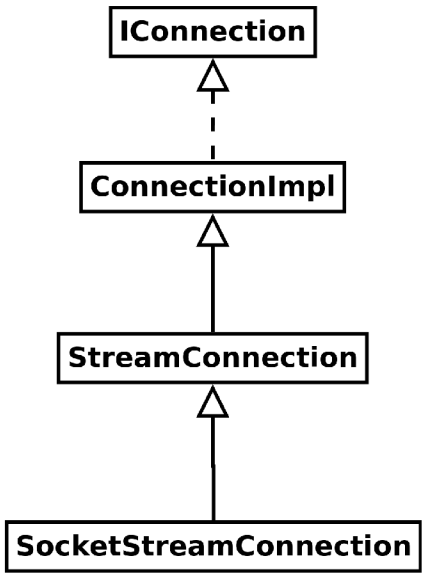
Spojení se může nacházet v jednom ze 3 stavů, jak je vidět na obr. 1.25.

Ve stavu STOPPED je spojení neaktivní a neprovádí žádnou činnost. Ve stavu



Obr. 1.25: Stavový diagram spojení

DISPATCHING spojení odebírá zprávy ze své vnitřní fronty zpráv a posílá je druhému ACP serveru a naopak od něj přijímá zprávy a předává je na zpracování registrovanému handleru. Vlastní mechanismus posílání zpráv řeší konkrétní třídy implementující rozhraní *IConnection*. Tyto třídy jsou členěny do hierarchie potomků, jak je vidět na obr. 1.26.



Obr. 1.26: Diagram tříd spojení

Rozhraní IConnection

Toto rozhraní je implementováno každým spojením ACP serveru. Obsahuje metody pro ovládání stavu spojení, pro přidávání zpráv do výstupní fronty a pro registraci handleru spojení.

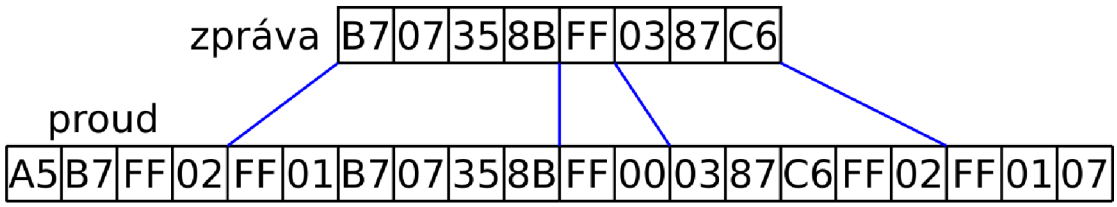
Třída ConnectionImpl

Abstraktní třída *ConnectionImpl* je obecná implementace rozhraní *IConnection*. Zajišťuje základní služby pro spojení - spravuje vlákna pro vysílání a příjem, frontu odchozích správ a registraci handleru spojení.

Třída StreamConnection

Tato abstraktní třída dědí ze třídy *ConnectionImpl* a využívá tak její funkcionality. Zajišťuje vkládání ACP zpráv do proudu na jedné straně spojení a jeho opětovné rozdělení na zprávy na straně druhé. Třída pracuje s obecným duplexním proudem dat, tvorbu konkrétního proudu zajišťují potomci této třídy.

Při vkládání zprávy do proudu je nutné označit její začátek a případně i konec. Označení začátku je nutné pro zajištění zprávové synchronizace. Spoléhat se na řazení zpráv za sebe není vhodné, protože při chybném příjmu zprávy by se přijímač nedokázal zasynchronizovat a přijmout žádnou další zprávu. Označit konec zprávy je vhodné proto, že spojení pak nemusí znát strukturu zprávy aby identifikovalo její konec a je proto schopno přenášet libovolnou sekvenci bytů.



Obr. 1.27: Mapování zpráv do proudu

Princip vkládání zpráv do proudu je vidět na obr. 1.27. Začátek zprávy je označen dvojbytovou značkou FF01 a konec zprávy je označen značkou FF02. Protože je možné, že výše uvedené značky budou obsaženy v přenášené zprávě, je nutné zprávu pozměnit tak, aby toto bylo vyloučeno. Proto jsou všechny výskyty bytu FF ve zprávě zaměněny za dvojbytovou značku FF00. Dekodér pak tuto značku opět nahradí původním bytem FF.

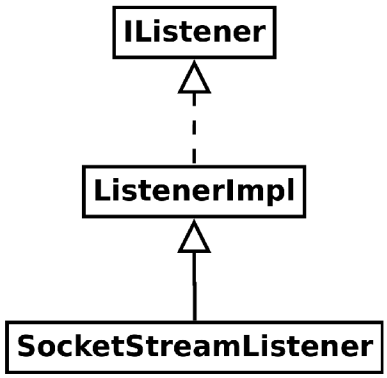
Třída SocketStreamConnection

Třída *SocketStreamConnection* je potomkem třídy *StreamConnection*, která jako vstupní a výstupní proud využívá socket. Socket představuje abstrakci poskytovanou prostředím Java pro síťovou komunikaci. Využil jsem návrhový vzor dependency

injection, tedy že konkrérní socket je předán třidě *SocketStreamConnection* v konstruktoru. Třída je tak na konkrétní implementaci socketu nezávislá. V programu se tato třída využívá jak pro čistý TCP proud, tak pro SSL/TLS nad TCP proudem.

1.7.6 Posluchači

Posluchač naslouchá příchozím spojením a pokud se někdo připojí, tak vytvoří odpovídající třídu implementující rozhraní *IConnection* a předá ji registrovanému handleru (tedy instanci třídy *Server*). Tyto třídy tvoří hierarchie potomků, jak je vidět na obr. 1.28.



Obr. 1.28: Diagram tříd posluchačů

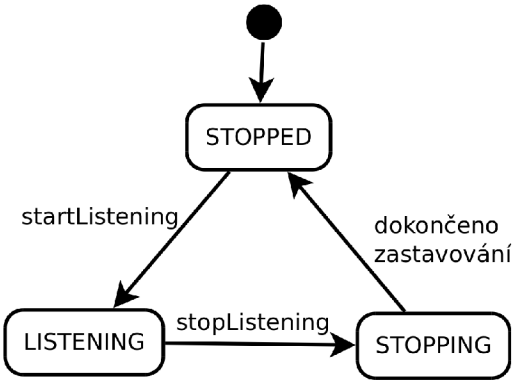
Rozhraní IListener

Každý posluchač implementuje rozhraní *IListener*. Toto rozhraní poskytuje abstrakci nad různými posluchači, jeho uživatelé tak nejsou závislí na konkrétním posluchači. Rozhraní poskytuje metody pro řízení svého stavu, pro nastavení svých parametrů a pro registraci seznamu spojení, do kterého budou vytvořená spojení přidávána (tedy instanci třídy *Server*). Posluchač se může nacházet v jednom ze 3 stavů, jak je vidět na obr. 1.29.

Po svém vytvoření je posluchač ve stavu STOPPED, ve kterém není aktivní. Po zavolání metody *startListening* přejde posluchač do stavu LISTENING, ve kterém naslouchá příchozím spojením a přidává je do zaregistrovaného seznamu spojení. Po zavolání metody *stopListening* přejde posluchač do stavu STOPPING a započne své zastavování. Jakmile je zastaven, tak přejde do výchozího stavu STOPPED.

Třída ListenerImpl

Třída *ListenerImpl* implementuje rozhraní *IListener*. Udržuje stav poslochače a dle něj spravuje vlákno, v němž je prováděno naslouchání. Také zajišťuje registraci se-



Obr. 1.29: Stavový diagram posluchače

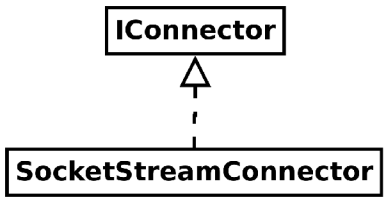
znamu spojení.

Třída SocketStreamListener

Třída *SocketStreamListener* je potomkem třídy *ListenerImpl* a nalouchá socketovým spojením na zadaném koncovém bodu. Koncový bod, na němž má posluchač naslouchat, je definován v nastavení předaném z vnějšku jako instance třídy *SocketStreamListenerSettings*, posluchač tedy není závislý na konkrétním typu socketu.

1.7.7 Spojovatelé

Spojovatel slouží k vytvoření spojení s nadřízeným ACP serverem. Na rozdíl od posluchače nemá spojovatel žádné vlákno, které by se trvale snažilo vytvořit spojení. Pokud se spojení nepodaří vytvořit ve stanoveném čase, tak vrátí chybu. Proto je spojovatel volán jako úloha z thread poolu serveru a pokud se spojení nepodaří, tak je tato úloha opět vložena do fronty. Stejně tak pokud dojde k rozpojení již vytvořeného spojení, tak server vloží spojovací úlohu do thread poolu. Hierarchie tříd je zakreslena na obr. 1.30.



Obr. 1.30: Diagram tříd spojovatelů

Rozhraní IConnector

Rozhraní *IConnector* představuje abstraktní rozhraní pro vytvoření spojení s odpovídajícím posluchačem v jiném ACP serveru. Třída *Server* proto není závislá na konkrétní implementaci spojovatele. Rozhraní obsahuje metody pro nastavení parametrů spojovatele a pro vytvoření spojení.

Třída SocketStreamConnector

Třída *SocketStreamConnector* je implementace spojovatele, která navazuje soketové spojení na adresu zadanou v nastavení koncovým bodem. Nastavení je předáváno z vnějšku jako instance třídy *SocketStreamListenerSettings*. Díky abstrakci poskytované prostředím Java lze tuto třídu použít jak pro navázání klasického TCP proudu, tak pro SSL/TLS nad TCP proudem.

1.7.8 Koncové body

SocketEndPoint

Abstraktní třída *SocketEndPoint* definuje adresu koncového bodu spojení. Protože tato třída implementuje rozhraní *IXmlPersistence*, tak je perzistentní a může být načítána a ukládána z/do konfiguračního souboru. Dále obsahuje metodu *connect*, která se pokusí vytvořit spojení s tímto koncovým bodem a vrátí spojený socket a metodu *createBoundSocket*, která vrátí serverový socket s nastavenou adresou tohoto koncového bodu. Uživatelé třídy *SocketEndPoint* (například posluchači a spojovatelé) tedy nemusí znát konkrétní typy koncových bodů, na které se napojují.

Třída InetSocketEndPoint

Třída *InetSocketEndPoint* představuje koncový bod TCP proudu. Na úrovni síťové vrstvy je podporován protokol IPv4 i IPv6. Třída si pamatuje adresu počítače (hostname nebo IP adresu) a port koncového bodu.

Třída SslSocketEndPoint

Třída *SslSocketEndPoint* představuje koncový bod SSL/TLS spojení vybudovaným nad TCP proudem. Členskou proměnnou třídy je instance třídy *InetSocketEndPoint* představující adresu podkladového TCP proudu.

1.7.9 Třída Message

Třída *Message* představuje zprávu protokolu ACP. Třída obsahuje typ zprávy, ID transakce (hodnotu pole Identifier) a seznam AVP. Třída je schopna zapsat svůj obsah do proudu bytů pomocí metody *writeToStream* a zpět z něj svůj obsah načíst pomocí metody *readFromStream*, přičemž formát se řídí specifikací protokolu ACP, viz [3].

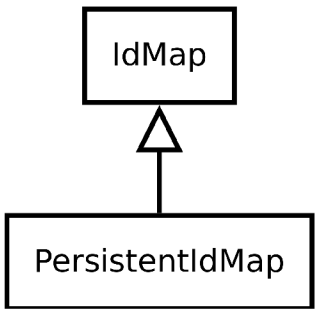
1.7.10 Třída Avp

Třída *Avp* obsahuje data jednoho AVP protokolu ACP. Je obsažena ve třídě *Message* pro reprezentaci obsahu jednotlivých AVP v seznamu. Proto se také musí být schopna načíst a uložit z/do proudu bytů. Třída explicitně rozpoznává pouze typ AVP, vlastní obsah interpretuje jen jako posloupnost bytů a jejich interpretace je pak ponechána na jiné vrstvě, konkrétně na skriptu.

1.7.11 Třídy IdMap a PersistentIdMap

Třída *IdMap* slouží k uchovávání objektů identifikovatelných pomocí číselného identifikátoru. Dokáže sama přidělovat identifikátory vloženým objektům tak, aby byly unikátní. Tato třída se používá pro uchovávání spojení a transakcí.

Potomkem třídy *IdMap* je třída *PersistentIdMap*, která akceptuje jen objekty implementující rozhraní *IXmlPersistence*. Třída umožňuje navíc uložit objekty do XML elementu a zpět je načíst. Používá se pro uchovávání spojovatelů a posluchačů. Diagram tříd je vidět na obr. 1.31.



Obr. 1.31: Diagram tříd IdMap a PersistentIdMap

1.7.12 Interpret JavaScriptu

Abych umožnil snadné experimentování s různými druhy autentizace a autorizace, tak jsem vlastní metody autentizace a autorizace vyčlenil mimo ACP server. Využívám implementaci skriptovacího jazyka JavaScript Rhino, protože je přímo obsažena

v JRE 1.6 a vyšším. Více informací o něm lze nalézt v [10]. Skript musí mít příponu .js, protože běhové prostředí jazyka Java podle něj rozpoznává typ skriptovacího jazyka a určuje, který interpret spustit. Skript může používat standardní knihovnu jazyka Java a může proto využívat její bohaté kryptografické funkce.

Pokud kořenový ACP server obdrží zprávu typu START a založí transakci, tak pro ni vytvoří vlastní instanci interpreteru a spustí uživatelem definovaný skript. Jednotlivé instance jsou tak od sebe odizolovány. Server nastaví skriptu proměnnou *server* na handler serveru - objekt implementující rozhraní *ITransactionHandle* a umožňující přístup skriptu k metodám serveru. Pokud server obdrží zprávu, tak příslušnému interpreteru zavolá funkci *onMessage* a zprávu jí předá spolu s handlerem transakce. Skript tak může na zprávu reagovat například odesláním odpovědi.

ACP server typu CLIENT transakci iniciuje, proto musí mít možnost odeslat první zprávu. Zároveň je třeba, aby uživatel měl možnost zadat identifikaci požadovaného aktiva a autentizační údaje. Proto má uživatel možnost vytvořit nový interpret a spustit v něm libovolný příkaz.

Preprocesor

Interpret JavaScriptu Rhino bohužel neumožňuje spuštěnému skriptu volat funkce z jiného souboru. Ve skriptu běžícím na kořenovém serveru i klientovy jsou společné funkce. Abych je nemusel mezi skripty kopírovat, tak jsem vytvořil preprocesor, který zpracuje skript před tím, než jej předá interpreteru. V současnosti má implementovaný jediný příkaz *include*, který vloží obsah jiného souboru na místo, kde se příkaz nachází. Vložený soubor je také zpracován preprocesorem a může proto inkludovat další soubory. Preprocesor je implementován ve třídě *Preprocessor*.

Třída ScriptRunner

Třída *ScriptRunner* má na starosti obsluhu skriptovacího enginu. Vytvoří jej, spustí v něm definovaný skript a v jeho kontextu umožňuje spouštět uživatelem zadané příkazy a volat definované funkce (například *onMessage*).

2 ZÁVĚR

Po nastudování protokolu ACP jsem navrhnul koncept implementace testovacího software. Tento software jsem vyvinul a důkladně otestoval. Dále jsem navrhnul a implementoval 3 testovací scénáře umožňující prověřit vlastnosti protokolu ACP i testovacího software. Nakonec jsem vypracoval podrobný návod k obsluze včetně vzorového příkladu. Tento je uveden v příloze diplomové práce.

Vyvinutý testovací software umožňuje důkladně otestovat protokol ACP a jeho vlastnosti. Je modulární a proto ho lze snadno rozšiřovat. Software podporuje síťové protokoly TCP/IP a TLS. Umožňuje snadno vyvinout různé metody řízení přístupu k aktivům a prakticky otestovat jejich funkčnost. Lze vytvořit rozsáhlejší síť ACP serverů a otestovat tak vlastnosti protokolu při současném přístupu mnoha žadatelů. Program obsahuje přívětivé uživatelské rozhraní umožňující jeho snadné ovládání. Přenositelnost software je zajištěna použitím programovacího jazyka Java.

Software jsem testoval v domácí síti na dvou počítačích. Stolní počítač byl připojen k routeru ethernetem 100Mbit/s, notebook pomocí bezdrátového Wi-Fi spoje. Zkoušel jsem různé konfigurace, od jednoduchého dvojbodového spoje mezi dvěma ACP servery až po víceúrovňovou hierarchii ACP severů, kde byly použity různé druhy přenosových kanálů. Ve všech případech se software choval správně.

Do budoucna je možné rozšířit přenosové protokoly například o nespojovaný síťový protokol UDP nebo využít rozhraní USB. Také je možné rozšířit strukturu sítě ACP serverů, aby bylo možné realizovat jinou topologii než strom. K tomu je nutné zavést adresování uzlů v síti a směrování zpráv Start. Také by bylo vhodné rozšířit software o možnost testování chybových stavů. Například by mohla jít nastavit pravděpodobnost zahození zprávy nebo bitová chybovost přenosu. Tím by šla testovat odolnost navržených testovacích scénářů proti chybám transportní vrstvy.

Dále by jistě bylo zajímavé zkusit propojit tento testovací software s ostatními implementacemi protokolu ACP vyvinutými na akademické půdě VUT. Některé aspekty komunikace však nejsou doposud standardizovány, například mapování zpráv ACP do TCP proudu, proto by bylo zřejmě nutné jednotlivé implementace upravit, aby byly vzájemně kompatibilní.

LITERATURA

[1] BURDA, Karel. *Bezpečnost informačních systémů.* Brno : FEKT Vysokého učení technického v Brně, 1.11.2005. 104 s.

[2] RFC 4252. *The Secure Shell (SSH) Authentication Protocol.* [s.l.] : The Internet Engineering Task Force, January 2006. 17 s. Dostupné z WWW: <http://www.ietf.org/rfc/rfc4252.txt>.

[3] RFC. *Access Control Protocol (ACP).* Brno : VUT Brno, October 26, 2011. 22 s.

[4] BURDA, Karel. *Univerzální rámec pro řízení přístupu v počítačových sítích.* Elektrorevue [online]. 8. 3. 2011, 2011, 9, [cit. 2011-12-11]. Dostupný z WWW: <http://elektrorevue.cz/cz/clanky/komunikacni-technologie/0/univerzalni-ramec-pro-rizeni-pristupu-v-pocitacovych-sitich/>. ISSN 1213-1539.

[5] Internet-Draft. *Access Control Protocol (ACP).* K. Burda, I. Strasil, T. Pelka, P. Stancik. [s.l.] : The Internet Engineering Task Force, December 5, 2011. 25 s. Dostupné z WWW: <https://datatracker.ietf.org/doc/draft-kaaps-acp/>.

[6] RFC 3748. *Extensible Authentication Protocol (EAP).* [s.l.] : Network Working Group, June 2004. 67 s. Dostupné z WWW: <http://www.ietf.org/rfc/rfc3748.txt>.

[7] Oracle. *Java™ Platform, Standard Edition 7 : API Specification* [online]. 2011 [cit. 2011-12-01]. Dostupné z WWW: <http://docs.oracle.com/javase/7/docs/api/index.html>.

[8] MCCONNELL, Steve. *Dokonalý kód : umění programování a techniky tvorby software.* Dotisk prvního vydání. Brno : Computer Press, 2006. 894 s. ISBN 80-251-0849-X.

[9] MATTHEW, Neil; STONES, Richard. *Linux : začínáme programovat.* Vydání první. Praha : Computer Press, 2000. 897 s. ISBN 80-7226-307-2.

[10] *Rhino : JavaScript for Java* [online].Mozilla, ©1998–2011, Last modified on December 11, 2011 [cit. 2011-12-11]. Dostupné z WWW: <http://www.mozilla.org/rhino/>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

AAA Authentication, authorization and accounting - autentizace, autorizace a účtování

ACP Access control protocol - univerzální protokol autentizace a autorizace

AES Advanced encryption standard - symetrická bloková šifra schválená úřadem NIST

API Application programming interface - rozhraní umožňující aplikaci přistupovat k funkcím softwarových komponent

AVP Attribute-value pair - datová struktura v rámci protokolu ACP nesoucí označené informace

CAVP Container AVP - AVP obsahující seznam jiných AVP

EAP Extensible authentication protocol - protokol autentizace definovaný v RFC 5247

IP Internet protocol - protokol síťové vrstvy definovaný v RFC 791

JRE Java runtime environment - běhové prostředí jazyka Java

LAVP Long AVP - dlouhá verze AVP

MD5 Message digest - hashovací funkce definovaná v RFC 1321

RSA Asymetrická šifra využitelná pro šifrování i elektronické podepisování

SAVP Short AVP - krátká verze AVP

SHA Secure hash algorithm - hashovací funkce navržená organizací NSA

SSH Secure shell - zabezpečená varianta programu Telnet

SSL Secure socket layer - protokol postavený nad transportní vrstvou zajišťující bezpečnou komunikaci po internetu

TCP Transmission control protocol - protokol poskytující spolehlivou transportní vrstvu definovaný v RFC 793

TLS Transport layer security - nástupce protokolu SSL

VSA Variant of single answer - defaultní varianta protokolu ACP

SEZNAM PŘÍLOH

A MANUÁL K ACP SERVERU

A.1 Inslalace a spuštění

Program není třeba nijak instalovat. Stačí zkopírovat soubor acp.jar do libovolného pracovního adresáře a spustit ho. Tento pracovní adresář pak lze přednostně použít pro ukládání konfiguračních souborů, klíčů a skriptů. Pro běh programu je nutné mít nainstalováno běhové prostředí Java Runtime Environment verze 1.6 nebo vyšší.

A.1.1 Argumenty programu

ACP server se z příkazové řádky spustí příkazem:
`java` *`argumenty běhového prostředí`* `-jar acp.jar` *`soubor s nastavením`*

Pokud není cesta k souboru s nastavením uvedena, tak je defaulně uvažován soubor settings.xml. Soubor nemusí před spuštěním ACP serveru existovat, pokud neexistuje, tak si jej server sám vytvoří. Možnost nastavení konfiguračního souboru zjednodušuje testování protokolu ACP na jednom počítači, kdy je nutné spustit několik instancí programu s různými konfiguracemi.

Možné argumenty běhového prostředí lze najít v [7]. Zmíním zde jen argumenty nutné pro nastavení parametrů SSL/TLS spoje. Na straně klienta podřízeného serveru je nutné mít nainstalovaný certifikát nadřízeného serveru, který je uložen v úložišti (trust store). Cestu k úložišti určuje argument:
`-Djavax.net.ssl.trustStore=cesta`

Na straně nadřízeného serveru je nutné mít nastavenou cestu k úložišti se soukromým klíčem. Tento klíč je vhodný mít šifrovaný heslem. Běhovému prostředí je pak třeba sdělit parametry:
`-Djavax.net.ssl.keyStore=cesta -Djavax.net.ssl.keyStorePassword=heslo`

Klíč i certifikát je možné mít uložený v jednom úložišti, například při provozování obou ACP serverů na stajném počítači. Klíč a certifikát lze vygenerovat pomocí programu keytool dodávaného spolu s běhovým prostředím Javy.

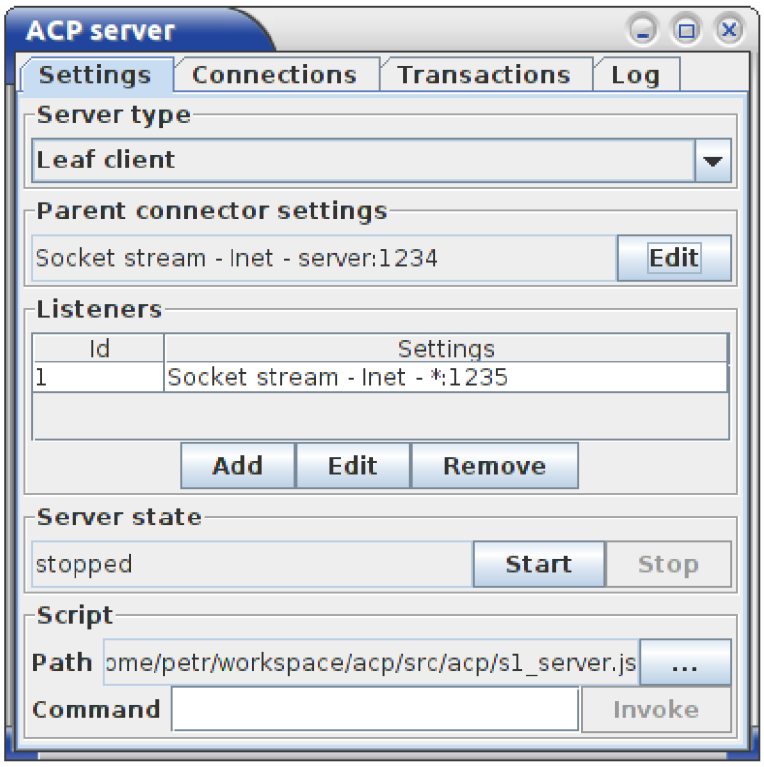
A.2 Ovládání

Okno ACP serveru sestává ze 4 záložek - settings, connections, transactions a log.

A.2.1 Záložka settings

Tato záložka umožňuje konfiguraci ACP serveru a řízení jeho stavu. Je logicky členěna tak, že se nastavování provádí odshora dolů.

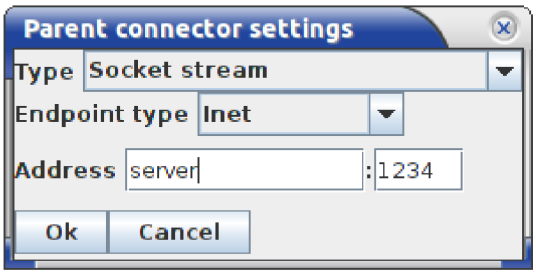
Nejdříve je nutné nastavit typ ACP serveru. K tomu slouží sekce Server type. Povolené hodnoty jsou:



Obr. A.1: Záložka settings

- Root server - kořenový ACP server provádějící autentizaci žadatelů a spravující aktiva
- Transit server - server přeposílající zprávy mezi jinými ACP servery
- Leaf klient - ACP server simulující klienta žadatele

V sekci Parent connector settings lze nastavit spojovatele navazujícího spojení na nadřízený ACP server. Toto nastavení není dostupné, pokud je server nastavený jako kořenový. V hlavním okně aplikace je zobrazeno aktuální nastavení spojovatele. Stisknutím tlačítka Edit se otevře dialog umožňující nastavení změnit, jak je vidět na obr. A.2.



Obr. A.2: Parent connector settings dialog

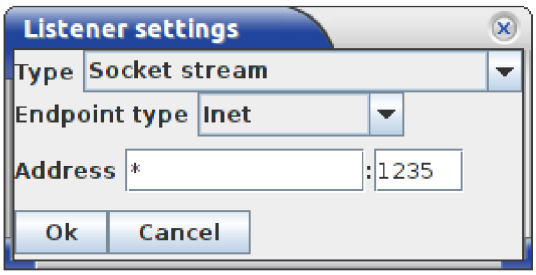
Seznam Type umožňuje nastavit typ spojovatele. V současnosti je dostupná jen

volba Socket stream představující spojení pomocí proudově orientovaných socketů. Tento spojovatel se umí připojit k více typům koncových bodů, které se nastavují v poli Endpoint type. Podporované typy koncových bodů jsou:

- Inet - TCP proud
- SSL/TLS - Šifrovaný spoj vybudovaný nad TCP proudem

Dále se nastavuje adresa a port tohoto koncového bodu. Adresu lze zadat jako hostname nebo IP adresu.

V sekci Listeners se nastavují posluchači naslouchající příchozím spojením od podřízených ACP serverů. Těchto posluchačů může být více a proto jsou zobrazeny v tabulce. Pomocí tlačítek Add, Edit a Remove lze tyto posluchače upravovat. Úpravy se provádí v dialogu Listener settings, jak je vidět na obr. A.3. Nastavení posluchače musí být stejné jako nastavení spojovatele podřízeného ACP serveru. Adresu lze vybrat jednu konkrétní, na které má posluchač naslouchat, nebo zadat "*"pro naslouchání na všech dostupných adresách.



Obr. A.3: Listener settings dialog

Sekce Server state slouží pro spuštění a zastavení běhu serveru. Pokud je server spuštěný, tak nelze měnit jeho konfiguraci.

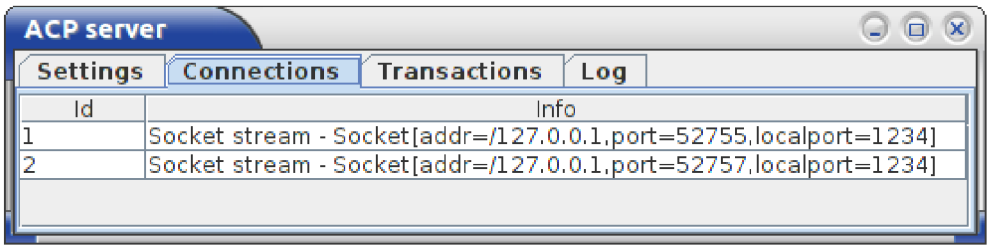
V sekci Script lze nastavit cestu ke skriptu, který je spuštěn po vytvoření transakce. Dále lze do pole Command zadat libovolný příkaz a provést ho v kontextu skriptu. Tímto způsobem lze vytvořit transakci v ACP serveru žadatele.

A.2.2 Záložka connections

Tato záložka obsahuje seznam spojení jejichž je ACP server účastníkem. Její vzhled je vidět na obr. A.4. O každém spojení je zobrazeno jeho ID, informace o jeho typu a adresa protějšího ACP serveru.

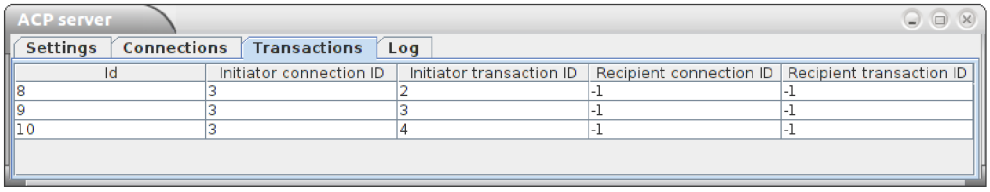
A.2.3 Záložka transactions

Tato záložka zobrazuje aktuálně probíhající transakce, jak je vidět na obr. A.5. Id představuje ID transakce v rámci serveru. Initiator connection ID představuje ID



Obr. A.4: Záložka connections

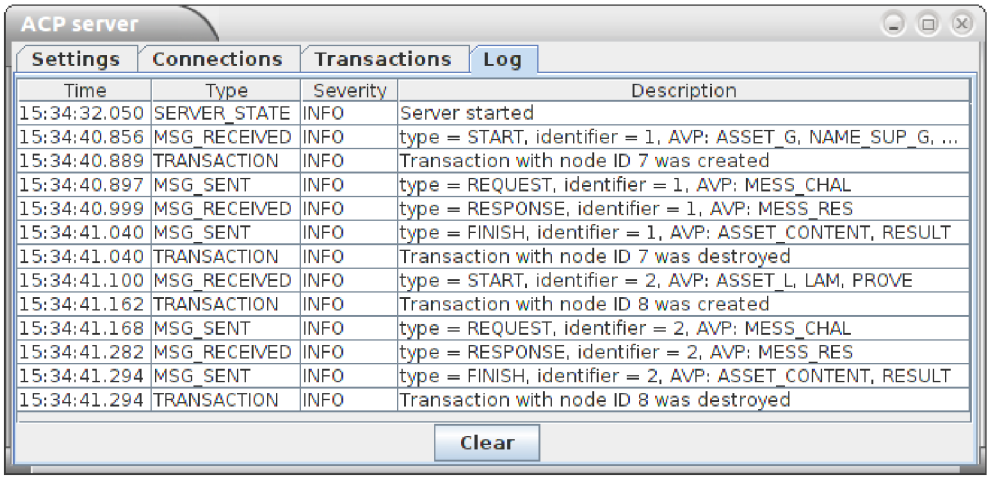
spojení od od iniciátora a Initiator transaction ID hodnotu pole Identifier v tomto spojení. Podobně, Recipient connection ID představuje ID spojení k poskytovateli a Recipient transaction ID a hodnotu pole Identifier v tomto spojení.



Obr. A.5: Záložka transactions

A.2.4 Záložka log

Zde jsou zobrazeny informace o událostech, které nastaly během běhu serveru. Její vzhled je vidět na obr. A.6. Jedná se o přijaté a odeslané zprávy, dále informace o navázání a zrušení transakcí, změnách stavu serveru a zprávy vypsané z logu.



Obr. A.6: Záložka log

A.3 Programování skriptů

Skripty jsou programovány v jazyce JavaScript a jsou interpretovány vestavěným enginem Rhino. Musejí mít příponu .js. Prováděný skript je nejdříve zpracován vestavěným preprocessorem, který umožňuje sestavit sktipr z více souborů.

Preprocessor na místo, kde je uveden příkaz

```
#include "nazev_souboru.js"
```

vloží obsah uvedeného souboru.

Popis jazyka JavaScript je nad rámec tohoto manuálu, více lze najít např. v [10].

A.3.1 API ACP serveru

ACP server poskytuje aplikační programové rozhraní, které lze využívat ve skriptech. Toto rozhraní umožňuje reagovat na události serveru, řídit transakce a odesílat v nich zprávy. Pro každou transakci je vytvořena nová instance skriptovacího enginu, pokud tedy ACP server zpracovává více transakcí, tak se tyto nijak vzájemně neovlivňují.

Žadatel vytvoří transakci zavoláním libovolného příkazu v kontextu skriptu. Toto se provádí v sekci Script. Pokud ACP server obdrží zprávu, tak odpovídajícímu skriptu zavolá funkci:

```
void onMessage (
final ITransactionHandle transaction, final Message receivedMessage)
```

Rozhraní IServerHandle

Toto rozhraní umožňuje přistupovat k funkcím ACP serveru jako celku. Reference na toto rozhraní je uložena v globální proměnné *server* dostupné ze skriptu.

Rozhraní deklaruje tyto metody:

- `ITransactionHandle createTransaction()` - metoda vytvoří novou transakci a vrátí na ni handle. Ten pak lze využít pro přístup k této transakci.
- `void destroyTransaction (final ITransactionHandle handle)` - metoda zruší předanou transakci.
- `Message createMessage()` - metoda vytvoří a vrátí novou zprávu ACP.
- `Avp createAvp()` - metoda vytvoří a vrátí nový AVP.

Rozhraní ITransactionHandle

Toto rozhraní slouží pro ovládání transakce. Skript získá jeho referenci při vytvoření transakce pomocí metody *IServerHandle.createTransaction()* nebo jako parametr funkce *onMessage*.

Rozhraní deklaruje tyto metody:

- `IServerHandle getServerHandle()` - metoda vrátí handle serveru, který vlastní transakci.
- `void sendMessage (final Message message)` - metoda odešle zadanou zprávu v rámci této transakce protistraně.

Třída Message

Tato třída reprezentuje jednu zprávu protokolu ACP. Obsahuje tyto metody:

- `MessageType getMessageType()` - vrátí typ zprávy jako enumeraci *MessageType*.
- `void setMessageType (final MessageType messageType)` - nastaví typ zprávy jako enumeraci *MessageType*.
- `int getConnectionTransactionId()` - vrátí hodnotu *Identifier*.
- `void setConnectionTransactionId (final int connectionTransactionId)` - nastaví hodnotu *Identifier*.
- `int getAvpCount()` - vrátí počet AVP ve zprávě.
- `Avp getAvpAt (final int index)` - vrátí AVP na dané pozici.
- `void addAvp (final Avp avp)` - přidá AVP na konec seznamu.
- `Avp findAvpWithType (final AvpType type)` - najde první AVP s daným typem.

Třída Avp

Tato třída reprezentuje jedno AVP protokolu ACP. Obsahuje tyto metody:

- `AvpType getType()` - vrátí typ AVP jako enumeraci *AvpType*.
- `void setType (final AvpType type)` - nastaví typ AVP jako enumeraci *AvpType*.
- `byte[] getValue()` - vrátí pole bytů s obsahem AVP.
- `void setValue(final byte[] value)` - nastaví obsah AVP.

Enumerace MessageType

Tato enumerace obsahuje typy zpráv. Možné jsou následující hodnoty představující stejnojmenné typy zpráv:
`MESSAGE_TYPE_START, MESSAGE_TYPE_OFFER, MESSAGE_TYPE_RESPONSE, MESSAGE_TYPE_REQUEST, MESSAGE_TYPE_SPECIFY, MESSAGE_TYPE_FINISH`

Význam hodnot lze nalézt v [3].

Enumerace AvpType

Tato enumerace obsahuje typy AVP. Možné jsou následující hodnoty představující stejnojmenné typy AVP:

AVP_NAME_SUP_G, AVP_NAME_PRO_G, AVP_NAME_AUT_G, AVP_NAME_ACC_G,
AVP_NAME_SUP_L, AVP_NAME_PRO_L, AVP_NAME_AUT_L, AVP_NAME_ACC_L,
AVP_ADDR_SUP_G, AVP_ADDR_PRO_G, AVP_ADDR_AUT_G, AVP_ADDR_ACC_G,
AVP_ADDR_SUP_L, AVP_ADDR_PRO_L, AVP_ADDR_AUT_L, AVP_ADDR_ACC_L,
AVP_EAP, AVP_LAM, AVP_GVP, AVP_LVP, AVP_ASSET_G, AVP_ASSET_L,
AVP_RESULT, AVP_PROVE, AVP_VERIF, AVP_FORM_START, AVP_MESS_RADIUS,
AVP_MESS_DIAMETR, AVP_MESS_KERBER, AVP_AVP_RADIUS, AVP_AVP_DIAMETR,
AVP_INIT, AVP_PMS, AVP_CERT, AVP_AES, AVP_ENC, AVP_HMAC, AVP_MAC,
AVP_RSA, AVP_PSS, AVP_SIGCRYPT, AVP_TXT, AVP_EAP_TX, AVP_LAM_TX,
AVP_ASSET_G_TX, AVP_ASSET_L_TX, AVP_ASSET_CONTENT, AVP_PUB_KEY_SUP,
AVP_MESS_CHAL, AVP_MESS_RES, AVP_TEST

Význam hodnot lze nalézt v [3].

Poznámky na závěr

Ve skriptu lze využívat třídy ze standardní knihovny jazyka Java. Jedná se zejména o různé kolekce, kryprografické funkce a podobně. Některé třídy jsou implementovány jak v JavaScriptu, tak v Javě. Je však mít na zřeteli, že tyto třídy jsou různé a nelze je zaměňovat. Jedná se zejména o řetězce a pole. Například převod řetězce java.lang.String na řetězec v JavaScriptu lze provést konstrukcí ” + *str*.

A.4 Příklad použití

V této sekci je popsán příklad tvorby jednoduchého skriptu a jeho spuštění. Jedná se o první testovací scénář popsaný v diplomové práci.

Nejdříve vytvoříme klientský skript client.js a uložíme ho do pracovního adresáře ACP serveru. Skript využívá společné knihovny common.js, proto ji inkluduje. Dále využívá kryptografického balíčku java.security a grafické knihovny javax.swing.

```
#include "common.js"

importPackage(java.security);
importPackage(javax.swing);
```

Deklarujeme proměnné pro uložení uživatelského jména, hesla a kódu aktiva.

```
var userName;
var password;
var asset;
```

Dále naprogramujeme funkci, která vyšle zprávu Start. Tato funkce je volána přímo žadatelem, na rozdíl od ostatních funkcích vyvolaných jako reakci na zprávu

ACP. Funkce nejdříve uloží žadatelem zadané údaje do globálních proměnných a vytvoří transakci. Dále vytvoří zprávu Start, vyplní v ní AVP typu ASSET_L, LAM a NAME_SUP_G a zprávu odešle v rámci vytvořené transakce.

```
function request(userName_, password_, asset_) {
  userName = userName_;
  password = password_;
  asset = asset_;

  var transaction = server.createTransaction();

  message = server.createMessage();
  message.setMessageType (MESSAGE_TYPE_START);

  var avpAsset = server.createAvp();
  avpAsset.setType (AVP_ASSET_L);
  avpAsset.setValue (byteToArray (asset));

  message.addAvp (avpAsset);

  var avpLam = server.createAvp();
  avpLam.setType (AVP_LAM);
  avpLam.setValue (byteToArray (LAM_PASSWORD_CHALLENGE));
  message.addAvp (avpLam);

  var avpUserName = server.createAvp();
  avpUserName.setType (AVP_NAME_SUP_G);
  avpUserName.setValue (stringToBytes (userName));

  message.addAvp (avpUserName);

  transaction.sendMessage(message);
}
```

Dále je nutná funkce reagující na zprávu ACP. Funkce musí mít jméno *onMessage* a uvedené parametry. Po svém zavolání funkce dle typu zprávy zavolá jinou funkci reagující na daný typ (*processRequest*, *processOffer* a *processFinish*). Tyto funkce jsou naprogramovány dále. Pokud typ zprávy není známý, tak je transakce zrušena.

```
function onMessage (transaction, receivedMessage) {
  var messageType = receivedMessage.getMessageType();

  switch (messageType) {
    case MESSAGE_TYPE_REQUEST:
      processRequest (transaction, receivedMessage);
      break;
```

```
    case MESSAGE_TYPE_OFFER:
      processOffer (transaction, receivedMessage);
      break;

    case MESSAGE_TYPE_FINISH:
      processFinish (transaction, receivedMessage);
      break;

    default:
      server.destroyTransaction (transaction);
      break;
  }
}
```

Funkce *processRequest* zpracovává zprávu typu Request. Nejdříve si ze zprávy přečte AVP typu MESS_CHAL a získá tak pole bytů - autentizační výzvu. Pak zavolá funkci *getPasswordResponse* které předá uložené heslo a přečtenou výzvu a funkce spočítá autentizační odpověď. Dále vytvoří AVP typu MESS_RES, vyplní do něj spočítanou autentizační odpověď a toto AVP vloží do nově vytvořené zprávy typu Response. Zprávu pak odešle v transakci.

```
function processRequest (transaction, receivedMessage) {
  // Process received message
  var avpChallenge = receivedMessage.findAvpWithType (AVP_MESS_CHAL);
  var challenge = avpChallenge.getValue();

  var response = getPasswordResponse (password, challenge);

  // Send message
  var responseAvp = server.createAvp();

  responseAvp.setType (AVP_MESS_RES);
  responseAvp.setValue (response);

  var messageToSend = server.createMessage();
  messageToSend.setMessageType (MESSAGE_TYPE_RESPONSE);
  messageToSend.addAvp (responseAvp);

  transaction.sendMessage (messageToSend);
}
```

Funkce *processFinish* zpracovává zprávu typu Finish. Z předané zprávy nejdříve získá AVP typu RESULT. Zkontroluje, že toto AVP obsahuje jednobytovou hodnotu RESULT_OK. Pokud ano, tak přečte AVP typu ASSET_CONTENT a zobrazí v něm obsažený řetězec v message boxu. Jinak zobrazí hlášku o chybě.

```
function processFinish (transaction, receivedMessage) {
  var avpResult = receivedMessage.findAvpWithType (AVP_RESULT);
  var result = avpResult.getValue();

  if (result.length == 1 && result[0] == RESULT_OK) {
    var avpAsset = receivedMessage.findAvpWithType (AVP_ASSET_CONTENT);
    var asset = bytesToString (avpAsset.getValue());

    JOptionPane.showMessageDialog(null, 'Received asset: ' + asset);
  }
  else {
    JOptionPane.showMessageDialog(null, 'Supplier returned error');
  }
}
```

Funkce *test1* slouží pro usnadnění testování, aby nebylo nutné vždy volat funkci *request* a předávat jí uživatelské jméno, heslo a kód aktiva.

```
function test1() {
  request ('petr', 'kocka', 2);
}
```

Dále musíme naprogramovat serverový skript - pojmenujeme ho server.js. Podobně jako v klientském skriptu inkludujeme knihovnu common.js a standardní knihovny z Javy.

```
#include "common.js"


importPackage(java.util);
importPackage(java.security);
```

Deklarujeme pole aktiv. Aktiva jsou identifikována kódem rovnajícímu se jejich indexu v poli počínaje indexem 0.

```
var assetArray = new Array(
  'Znicte palebne prostredky nepritele',
  'Provedte utok na letadlovou lod',
  'Vypustte jaderne strely'
);
```

Dále deklarujeme mapu uživatelských jmen a hesel. Klíčem v mapě je uživatelské jméno, hodnotou odpovídající heslo.

```
var passwordMap = new HashMap();
passwordMap.put('petr', 'kocka');
passwordMap.put('pavel', '1234');
passwordMap.put('franta', 'rock');
```

Vytvoříme kryptograficky silný generátor náhodných čísel. Ten bude použit pro generování autentizační výzvy. Konstanta *CHALLENGE_SIZE* určuje délku výzvy v bytech.

```
var rng = new SecureRandom();
var CHALLENGE_SIZE = 16;
```

Dále deklarujeme globální proměnné pro uložení kódu požadovaného aktiva, kódu autentizační metody, uživatelského jména a vygenerované výzvy.

```
var requestedAsset;
var requestedLam;
var userName;
var challengeSent;
```

Dále je nutná funkce reagující na zprávu ACP. Funkce musí mít jméno *onMessage* a uvedené parametry. Po zavolání funkce dle typu zprávy zavolá jinou funkci reagující na daný typ (*processStart* a *processResponse*). Tyto funkce jsou naprogramovány dále. Pokud typ zprávy není známý, tak je transakce zrušena.

```
function onMessage (transaction, receivedMessage) {
  var messageType = receivedMessage.getMessageType();

  switch (messageType) {
    case MESSAGE_TYPE_START:
      processStart (transaction, receivedMessage);
      break;

    case MESSAGE_TYPE_RESPONSE:
      processResponse (transaction, receivedMessage);
      break;

    default:
      server.destroyTransaction (transaction);
      break;
  }
}
```

Metoda *processStart* zpracuje zprávu Start. Nejdříve z AVP typu NAME_SUP_G načte uživatelské jméno, z AVP typu ASSET_L kód požadovaného aktiva a z AVP typu LAM kód autentizační metody. Dále pak vygeneruje náhodnou výzvu s využitím dříve vytvořeného generátoru náhodných čísel, novou zprávu, vytvoří AVP typu MESS_CHAL a výyvu do něj vloží. Vytvoří novou ACP zprávu, vloží do ní AVP a zprávu odešle v transakci.

```
function processStart (transaction, receivedMessage) {
  // Process received message
```

```
  var avpUser = receivedMessage.findAvpWithType (AVP_NAME_SUP_G);
  userName = bytesToString(avpUser.getValue());

  var avpAsset = receivedMessage.findAvpWithType (AVP_ASSET_L);
  var assetValue = avpAsset.getValue();

  if (assetValue.length == 1)
    requestedAsset = assetValue[0];

  var avpLam = receivedMessage.findAvpWithType (AVP_LAM);
  var lamValue = avpLam.getValue();

  if (lamValue.length == 1) {
    requestedLam = lamValue[0];
  }

  // Send request
  challengeSent = newByteArray (CHALLENGE_SIZE);
  rng.nextBytes (challengeSent);

  var avpChallenge = server.createAvp();
  avpChallenge.setType (AVP_MESS_CHAL);
  avpChallenge.setValue (challengeSent);

  var messageToSend = server.createMessage();
  messageToSend.setMessageType (MESSAGE_TYPE_REQUEST);
  messageToSend.addAvp (avpChallenge);

  transaction.sendMessage (messageToSend);
}
```

Metoda *processResponse* zpracovává zprávu Response. Nejdříve vyhledá AVP typu MESS_RES. Z něj metoda získá autentizační odpověď. Dále metoda vyhledá heslo žadatele v mapě *passwordMap*. Pomocí něj spočítá správnou odpověď žadatele - stejně jako žadatel - pomocí funkce *getPasswordResponse*. Následně metoda vytvoří zprávu typu Finish. Do ní vloží AVP typu RESULT. Pokud je autentizační odpověď žadatele stejná jako vypočítaná, tak bude hodnota AVP RESULT nastavena na konstantu RESULT_OK a do zprávy bude vloženo AVP typu ASSET_CONTENT. Toto AVP bude obsahovat požadované aktivum vyhledané pomocí svého kódu v poli *assetArray*. Pokud je odpověď chybná, tak je hodnota VP RESULT nastavena na konstantu RESULT_ERROR a žádné aktivum posíláno není.

```
function processResponse (transaction, receivedMessage) {
  // Process received message
  var avpResponse = receivedMessage.findAvpWithType (AVP_MESS_RES);
  var response = avpResponse.getValue();
```

```
  var password = '' + passwordMap.get (userName);
  var expectedResponse = getPasswordResponse (password, challengeSent);

  // Send message
  var messageToSend = server.createMessage();
  messageToSend.setMessageType (MESSAGE_TYPE_FINISH);

  var avpResult = server.createAvp();
  avpResult.setType (AVP_RESULT);

  if (Arrays.equals (response, expectedResponse)) {
    var asset = '' + assetArray[requestedAsset];
    var avpAssetContent = server.createAvp();

    avpAssetContent.setType (AVP_ASSET_CONTENT);
    avpAssetContent.setValue (stringToBytes (asset));

    messageToSend.addAvp (avpAssetContent);

    avpResult.setValue (byteToArray (RESULT_OK));
  }
  else {
    avpResult.setValue (byteToArray (RESULT_ERROR));
  }

  messageToSend.addAvp (avpResult);
  transaction.sendMessage (messageToSend);
}
```

Nakonec je nutné vytvořit knihovnu common.js. V tomto manuálu je popsána tvorba té části knihovny, která je využita v prvním testovacím scénáři.

Nejdříve importujeme standardní balíček java.security s kryptografickými funkcemi. Také definujeme konstanty pro AVP typu RESULT vyjadřující úspěch nebo neúspěch transakce.

```
importPackage (java.security);

var LAM_PASSWORD_CHALLENGE = 0;
var LAM_SIGNATURE_CHALLENGE = 1;

var RESULT_OK = 0;
var RESULT_ERROR = 2;
```

Také je nutné naprogramovat funkci *getPasswordResponse*, která vypočítá autentizační odpověď na základě předaného hesla (řetězce) a autentizační výzvy (pole

bytů). Funkce nejdříve získá instanci hešovací funkce SHA-256. Tuto funkci inicializuje (promaže její stav). Pak do ní vloží pole bytů hesla. K získání pole bytů z řetězce využije funkci *stringToBytes*. Následně do hešovací funkce zapíše výzvu a spočítá heš. Vrátí tak hodnotu SHA-256 (password | challenge).

```
function getPasswordResponse (password, challenge) {
  var md = MessageDigest.getInstance ("SHA-256");

  md.reset();
  md.update (stringToBytes (password));
  md.update (challenge);

  return md.digest();
}
```

Jak již bylo řečeno výše, funkce *stringToBytes* převádí řetězec na pole bytů. Funkce předpokládá, že řetězec je typu ASCII. Nejdříve zaalokuje pole bytů potřebné velikosti, pak postupně naplní jednotlivé byty pole hodnotami získanými metodou *charCodeAt* a nakonec pole vrátí.

```
function stringToBytes (str) {
  var bytes = newByteArray (str.length);

  for (i = 0; i < str.length; i++) {
    bytes[i] = str.charCodeAt (i);
  }

  return bytes;
}
```

Metoda *bytesToString* dělá přesný opak - převede pole bytů na řetezec jazyka Java. K řetězci postupně připojuje znaky získané funkcí *fromCharCode* z jednotlivých bytů.

```
function bytesToString (bytes) {
  var str = '';

  for (i = 0; i < bytes.length; i++) {
    str += String.fromCharCode (bytes[i]);
  }

  return str;
}
```

Protože alokace pole jazyka Java není úplně přímočará a toto pole je jiného typu než pole z JavaScriptu, tak je zde funkce *newByteArray*, která alokuje pole bytů zadané velikosti. K tomu využívá reflexi.

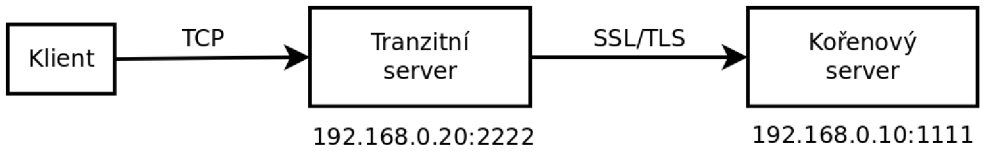
```
function newByteArray (length) {
  return Packages.java.lang.reflect.Array.newInstance (java.lang.Byte.
      TYPE, length);
}
```

Funkce *byteToArray* je pomocná funkce, která přebírá jeden byte jako argument a vrátí pole s tímto jedním bytem. Funkce se využívá pro nastavování hodnoty jednobytových AVP.

```
function byteToArray (value) {
  var array = newByteArray(1);
  array[0] = value;

  return array;
}
```

Nyní zprovozníme síť 3 ACP serverů. Schéma je vidět na obr.A.7.



Obr. A.7: Záložka log

Spoj mezi klientským AVP serverem a tranzitním serverem je TCP proud. Mezi tranzitním serverem a kořenovým serverem je TLS spoj. Proto je nutné vygenerovat klíč kořenového serveru. K tomu slouží program keytool dodávaný spolu s běhovým prostředím Javy.

Nejdříve vygenerujeme klíč serveru a uložíme ho do úložiště server_keystore. Program se zeptá na heslo úložiště, které vhodně zvolíme, a na heslo klíče, které ponecháme stejné jako heslo úložiště. Ostatní informace nemusíme vyplňovat (jméno vlastníka, adresu, ...).

```
keytool -genkeypair -keystore server_keystore
```

Z úložiště vyexportujeme certifikát serveru:

```
keytool -keystore server_keystore -exportcert -file server.cert
```

Certifikát přeneseme na počítač s tranzitním serevrem a importujeme ho do úložiště. Opět budeme dotázáni na heslo pro zajištění integrity uložených certifikátů.

```
keytool -keystore tranzit_keystore -importcert -file server.cert
```

Nyní přistoupíme ke spuštění 3 ACP serverů. Každý server bude mít nastavený svůj vlastní konfigurační soubor a kořenový a transitní serverver musí mít navíc přístup k uložišti klíčů.

Klientský server spustíme příkazem

`java -jar acp.jar setting_client.xml`

tranzitní server spustíme příkazem

`java -Djavax.net.ssl.trustStore=tranzit_keystore`
`-jar acp.jar setting_transit.xml`

a kořenový server příkazem

`java -Djavax.net.ssl.keyStore=server_keystore`
`-Djavax.net.ssl.keyStorePassword=123456 -jar acp.jar`
`setting_server.xml`

Místo `acp.jar` můžeme uvést cestu k programu, pokud se tento nenachází v aktuálním adresáři. Stejně tak můžeme soubory s nastavením (setting_client.xml, setting_transit.xml a setting_server.xml) umístit do jiného adresáře a uvést plnou cestu. Místo hesla 123456 pochopitelně uvedeme skutečné heslo do uložiště klíčů.

Dále servery musíme nakonfigurovat. Dejme tomu, že kořenový server bude umístěn na počítači s IP adresou 192.168.0.10 a přidělíme mu port 1111 a tranzitní server bude umístěn na počítači s IP adresou 192.168.0.20 a přidělíme mu port 2222. Toto je vidět na obr. A.7.

Nejdříve nastavíme typy serverů - kořenovému serveru nastavíme typ `Root server`, tranzitnímu serveru `Transit server` a klientskému serveru `Leaf client`.

Klientskému ACP serveru nastavíme spojovatele
`Socket stream - Inet - 192.168.0.20:2222`.

Tranzitnímu ACP serveru nastavíme posluchače
`Socket stream - Inet - 192.168.0.20:2222`

a spojovatele
`Socket stream - SSL - Inet - 192.168.0.10:1111`.

Kořenovému serveru nastavíme posluchače
`Socket stream - SSL - Inet - 192.168.0.10:1111`.

Dále nastavíme kořenovému serveru cestu k serverovému skriptu a kliensképu serveru cestu ke klientskému skriptu. Všechny servery spustíme a vykonáním příkazu `test1()` spustíme testovací scénář.