# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS


# VIRTUÁLNÍ PROSTŘEDÍ PŘÍSTUPU K UZLŮM V PLANETLAB

DIPLOMOVÁ PRÁCE
MASTER'S THESIS


AUTOR PRÁCE Bc. JIŘÍ FIC
AUTHOR



BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# VIRTUÁLNÍ PROSTŘEDÍ PŘÍSTUPU K UZLŮM V PLANETLAB

VIRTUAL ACCESS TO NODES IN PLANETLAB

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JIŘÍ FIC
AUTHOR

VEDOUCÍ PRÁCE — Ing. ONDŘEJ RYŠAVÝ, Ph.D.
SUPERVISOR

BRNO 2008

# Abstrakt

PlanetLab jako testovací platforma distribuovaných systémů nabízí jedinečnou příležitost pro vývoj a testování nových aplikací využitelných pro potřeby budoucího Internetu. Tato studie přináší návrh a řešení problému přístupu většího množství uživatelů – studentů k této platformě např. za účelem řešení školních projektů z této oblasti. Navržený systém umožňuje jeho správci vytvářet a kontrolovat virtuální uživatelské účty, díky nimž se mohou všichni jeho uživatelé připojovat na vybrané uzly PlanetLabu.



# Klíčová slova

PlanetLab, PlanetLab API, distribuovaný systém, překryvná síť, virtualizace, distribuovaná virtualizace, uzel, slice, virtuální server, SSH, SSH server, Linux, RSA



# Abstract

PlanetLab as a distributed systems testbed offers a unique opportunity for developing and testing new applications useful for future Internet. This work brings up a scheme and a solution of the problem with accessing PlanetLab by a larger group of students e.g. for the purpose of solving their courseworks. A designed system empowers its administrator to create and control virtual user accounts which provide possibility for all its users to connect to selected nodes in the PlanetLab.



# Keywords

PlanetLab, PlanetLab API, distributed system, overlay network, virtualization, distributed virtualization, node, slice, virtual server, SSH, SSH server, Linux, RSA



# Citace

Jiří Fic: Virtuální prostředí přístupu k uzlům
v PlanetLab, diplomová práce, Brno, FIT VUT v Brně, 2008

# Virtuální prostředí přístupu k uzlům v PlanetLab

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Ondřeje Ryšavého, Ph.D.

. . . . . .
Jiří Fic
19. května 2008

## Poděkování

Na tomto místě bych chtěl poděkovat svému vedoucímu Ing. Ondřeji Ryšavému, Ph.D. za odborné vedení a poskytnuté rady při vytváření této práce.

# Obsah

# Kapitola 1

# Úvod

## 1.1 Motivace

PlanetLab jako platforma pro testování softwaru distribuovaného charakteru v globálním měřítku nabízí řešení, jak výzkumným pracovníkům a lidem zabývajícím se vývojem distribuovaných aplikací umožnit a usnadnit proces vývoje a testování jejich aplikací.

Na této platformě vzniká mnoho významných projektů, jež mohou odstranit problémy současného Internetu v podobě propustnosti sítě, kvality služeb, bezpečnosti a mohou pomoci zvýšit jeho funkčnost.

V rámci České republiky se však zájem o tuto platformu zatím příliš neprojevil, v současné době je členem projektu pouze sdružení CESNET, díky němuž má k PlanetLabu přístup i VUT v Brně.

Poněvadž současný systém přístupu do PlanetLabu z pozice VUT v Brně není technicky dost dobře možný pro větší skupiny studentů, nabízí se řešení tohoto problému formou virtuálních účtů jako abstrakce nad existujícími účty.

Vytvořením virtuálních účtů pro přístup do PlanetLabu umožníme většímu množství studentů přistupovat k této platformě a vedoucím kurzů počítačových sítí tak nabídneme řešení, jak využít PlanetLab pro řešení studentských projektů z oblasti distribuovaných systémů.

## 1.2 Členění dokumentu

Kapitola 2 přináší informace o platformě PlanetLab, vyvíjených projektech, cílech a o architektuře jejích stěžejních částí souvisejících s přístupem uživatelů na ni. V dalších částech kapitoly je rozebráno programové rozhraní pro vzdálenou správu PlanetLabu – PlanetLab API a existující nástroje, které byly vyvinuty za účelem zjednodušení běžných správních činností.

Kapitola 3 vysvětluje důvody pro potřebu vytvoření systému na správu virtualizovaných účtů pro přístup do PlanetLabu a specifikuje požadavky na něj.

Stěžejní část celé práce představuje kapitola 4. Jsou zde obecně analyzovány dva možné způsoby řešení specifikovaného systému – systém na bázi SSH proxy serveru a systém dedikovaných SSH serverů. Druhý jmenovaný je poté detailněji analyzován a navržen. Představeny jsou jeho tři části, je vysvětlen princip rozdělení uživatelů do skupin, navržen systém řízení uzlů a jsou definovány principy zajištující bezpečnost přístupu.

Implementaci navrženého systému popisuje kapitola 5. Ukazuje využití PlanetLab API,

popisuje instalaci dedikovaných SSH serverů a správu virtuálních uživatelských účtů. Dále jsou představeny využité technologie XML-RPC a SSH, příkazy linuxového shellu, využité pro správu serverů, a nakonec je zhodnoceno zabezpečení jednotlivých částí systému.

V kapitole „Závěr“ jsou zhodnoceny dosažené výsledky a popsány možnosti rozšíření implementovaného systému.

Diplomová práce navazuje na semestrální projekt se shodným názvem, který si kladl za cíl porozumět platformě PlanetLab a specifikovat požadavky na systém, který by realizoval správu virtuálních uživatelských účtů.

# Kapitola 2

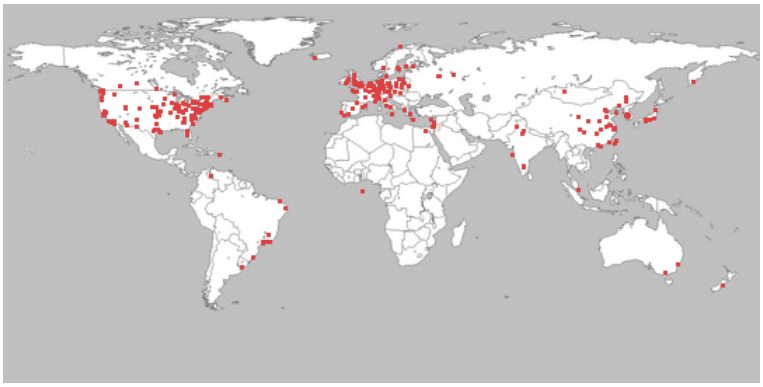
# PlanetLab

PlanetLab, jak již název napovídá, je celoplanetárně (globálně) distribuovaná překryvná počítačová síť, určená pro vývoj a testování distribuovaných aplikací. Dalo by se říci, že se jedná o jakousi laboratoř, inkubátor pro softwarové projekty. Za stejným názvem se také skrývá konsorcium [3], které celou tuto platformu vyvíjí a zaštiťuje.

## 2.1 Základní informace

Konsorcium bylo založeno roku 2002 jako společenství amerických univerzit (Princeton a Berkeley) a přidaly se i další univerzity, výzkumné týmy z komerční sféry (Intel, HP) a různá nadnárodní výzkumná pracoviště. Většina uzlů se nachází v USA, další pak zejména v Evropě a Jihovýchodní Asii. Zbytek je pak rozprostřen po celé planetě.

Většina počítačů tvořících PlanetLab je hostována výzkumnými institucemi, všechny jsou připojeny k Internetu. V současné době PlanetLab obsahuje 833 uzlů v 411 různých lokacích. Rozmístění těchto uzlů ilustruje obrázek 2.1. Cílem je zahrnout 1000 uzlů široce distribuovaných po celém světě, přičemž většina by měla být napojena na regionální či dálkové páteřní sítě.



Obrázek 2.1: Distribuované lokace PlanetLabu

Každá organizace, která se chce začlenit do PlanetLabu, nabídne k hostování několik svých serverů (uzlů) a na oplátku získá přístup ke sdíleným prostředkům v rámci celé platformy pro nasazení a testování svých projektů. Navíc, platforma sama o sobě, včetně

podstatných služeb potřebných k běhu, je navržena samotnou komunitou a je zajímavým tématem výzkumu.

Zásadním přínosem PlanetLabu je to, že uživatelé zde mohou vytvářet nezávislé aplikace, které mohou běžet současně tak, že se navzájem neovlivňují, a přitom využívají stejné uzly. Každý tak má k dispozici celý rozsah globální sítě. Mohou si tedy vytvářet ucelené virtuální vrstvy sítě, využívající stovky různě rozmístěných uzlů.

PlanetLab se neorientuje pouze na krátkodobé experimenty, nýbrž je navržen tak, aby podporoval také dlouhodobě běžící služby, tzn. na PlanetLab se nemusíme dívat pouze jako na testovací platformu, ale také jako na platformu pro nasazení, aplikace zde prochází fázemi od brzkého prototypu skrz mnoho návrhových iterací k populárním službám, které se nadále rozvíjejí.

Využití sítě obojím způsobem, jako testovací platformy a jako platformy pro nasazení, má své výhody. V prvém případě hodnota spočívá v tom, že uživatelé mají přístup k velkému množství geograficky vzdálených počítačů, k realisticky se chovající síti zažívající zahlcení, chyby a různorodé chování linek a realistickému vytížení testovaných služeb. V druhém případě je hodnotou pro výzkumníky přímá cesta vývoje populárních služeb a pro koncové uživatele přístup k těmto službám.

## 2.2 Projekty

Projekty, které jsou vyvíjeny v PlanetLabu, jsou orientovány zejména na služby širšího rozsahu. Mezi hlavní zaměření projektů patří:

- distribuovaná úložiště dat
- P2P systémy
- DHT (Distributed Hash Tables)
- překryvné sítě pro routování a multicast
- překryvné sítě pro QoS
- mapování sítí
- zpracování dotazů (query processing)

K nejznámějším vyvíjeným projektům patří např. projekt OceanStore [16], což je globální persistentní datový sklad navržený pro velkou škálu uživatelů (v řádech miliard), jehož hlavními přednostmi jsou konzistence, vysoká dostupnost a odolnost, který je vystavěn na vrcholu infrastruktury zahrnující nedůvěryhodné servery.

Dalšími významnými projekty jsou např. distribuční systém Coral [2] nebo distribuovaný DNS systém CoDNS [1].

## 2.3 Cíle

PlanetLab měl od počátku stanoveny tyto cíle [17, 18]:

- poskytovat platformu pro experimentování se síťovými službami celoplanetárního rozsahu

- poskytovat platformu pro nasazení a využívání nových síťových služeb, které by sloužily skutečné komunitě uživatelů
- podnítit vývoj Internetu na architekturu orientovanou na služby

Snahou je zajistit škálovatelnost, bezpečnost, robustnost a decentralizaci, kterou architektura Internetu vyžaduje, avšak musí být současně zajištěno, že tato architektura bude řízena požadavky existujícího systému a ne pouze jakousi idealizovanou představou.

Konsorcium PlanetLab je zodpovědné za dohlížení na dlouhodobý rozvoj hardwarové infrastruktury, rozvíjení své softwarové architektury, poskytování každodenní provozní podpory a definování politiky, která řídí odpovídající použití.

## 2.4 Terminologie

Protože problematika PlanetLabu přináší řadu specifických termínů, v této části jsou vysvětleny nejvýznamnější z nich.

**Lokace (site):** Geografické místo, kde jsou umístěny uzly PlanetLabu (např. Princetonská univerzita, CESNET). Zkrácené verze jejich názvů tvoří prefix názvu pro každý *slice* (např. `cesnet_vutbr2`).

**Uzel:** Dedikovaný server, na kterém běží komponenty služeb PlanetLabu.

**Slice:** Množina alokovaných zdrojů distribuovaných skrz PlanetLab. Pro většinu uživatelů znamená *slice* přístup přes unixový shell na uzly PlanetLabu.

**Sliver:** Množina alokovaných zdrojů na jednom uzlu PlanetLabu.

**Virtuální server (*vserver*):** *Slivery* jsou v současnosti implementovány jako linuxové *vservery* [5], které implementují izolaci jmenného prostoru a zdrojů mezi *slivery* na jednom uzlu. Síťová virtualizace *sliverů* je implementována pomocí VNETu [11]. Občas mohou být termíny *slice*, *sliver* a *vserver* zaměněny, ale existují mezi nimi architektonické rozdíly.

**Distribuovaná virtualizace:** Získání distribuované množiny virtuálních strojů, které jsou systémem považovány jako jediná složená entita.

**Správce (Principal Investigator – PI):** Správce lokace, je na dané lokaci zodpovědný za údržbu *sliců* a uživatelů. Správci jsou zodpovědní za chování svých *sliců*. Většina lokací má pouze jednoho správce – typicky zaměstnanec fakulty vysoké školy nebo projektový manažer v komerční instituci.

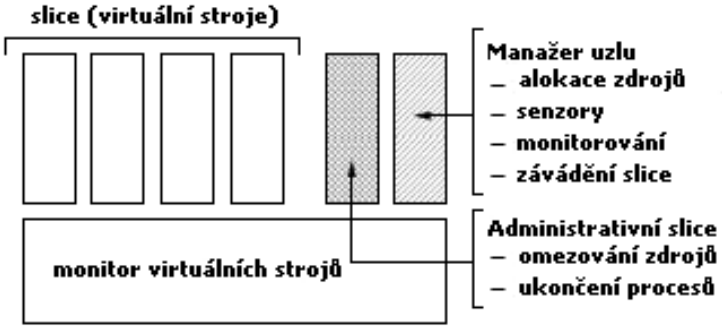
**Technický kontakt:** Subjekt zodpovědný za instalaci, údržbu a monitorování uzlů v lokaci. V každá lokaci existuje alespoň jeden.

**Uživatel:** Kdokoliv, kdo vyvíjí a nasazuje aplikace do PlanetLabu. Správci mohou být rovněž uživateli.

## 2.5 Architektura

### 2.5.1 Obecná struktura uzlu

Jako základ každého uzlu byl zvolen plnohodnotný operační systém – Linux, kvůli jeho rozšířenosti ve výzkumné komunitě s tím, že bude postupně upravován na základě zkušeností získaných za provozu. Jakási meta-architektura, podle které se ubíral vývoj, je zobrazena na obrázku 2.2




Obrázek 2.2: Architektura uzlu PlanetLabu

Na nejnižší úrovni na každém uzlu PlanetLabu běží monitor virtuálních strojů (Virtual Machine Monitor – VMM), který implementuje a izoluje virtuální stroje. VMM také definuje API, na němž jsou služby implementovány. V současnosti PlanetLab implementuje VMM jako kombinaci jádra Linuxu a jisté množiny rozšíření jádra.

Privilegovaný (root) virtuální stroj, běžící na vrcholu VMM, nazývaný *manažer uzlu* (node manager), monitoruje a spravuje veškeré virtuální stroje na uzlu, tzn. řídí jejich vytváření a přidělování prostředků pro ně. Existují zde také speciální *infrastrukturní slicy*, které vykonávají základní funkce na každém uzlu, např. poskytují rozhraní k uzlu pro lokální administrátory.

Některé služby běžící na vrcholu VMM mohou být určitým způsobem charakterizovány jako privilegované – mohou provádět privilegované volání manažera uzlu (kvůli alokaci prostředků). Avšak toto se týká pouze některých infrastrukturních služeb, uživatelské služby jsou všechny neprivilegované.

V současnosti existují tři typy infrastrukturních služeb:

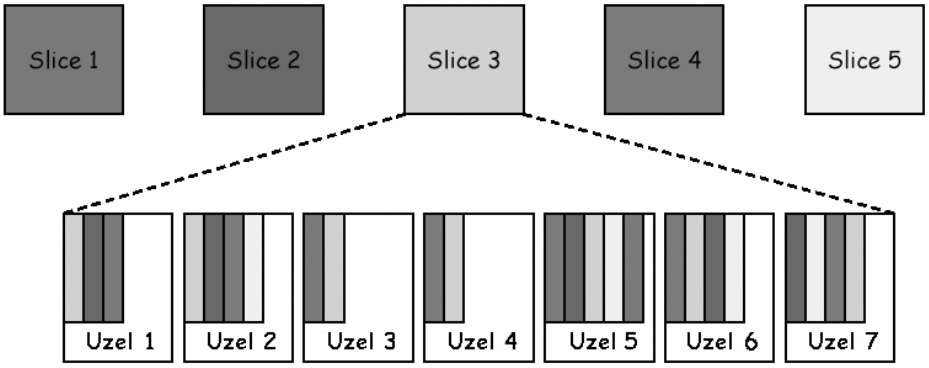
- jednatelské služby(brokerage services) – určeny pro získávání zdrojů a vytváření *sliců*
- služby prostředí – slouží k inicializaci a údržbě kódu *slice*
- monitorovací služby – určeny k zjišťování dostupných zdrojů a sledování stavu běžících služeb

Podrobnější informace lze nalézt v [14].

### 2.5.2 Slice

Služby a aplikace PlanetLabu běží v části platformy (*slice*) – množině uzlů, na kterých služba získá podíl zdrojů z každého uzlu, ve formě virtuálního stroje – viz obrázek 2.3.

PlanetLab zde zavádí pojem *distribuovaná virtualizace*, což znamená získání distribuované množiny virtuálních strojů, které jsou systémem považovány jako jediná složená entita.




Obrázek 2.3: *Slice* na množině uzlů

Za vytváření *sliců* a přiřazování uživatelů k nim je zodpovědný správce. Poté, co je uživatel přiřazen ke *slice*, může k němu přiřazovat uzly. Jakmile má *slice* přiřazeny uzly, vytvoří se na každém z nich virtuální server pro daný *slice*. Každý *slice* má konečnou dobu platnosti (obvykle 2 měsíce) a jejich platnost musí být periodicky obnovována pro jejich zachování.

### 2.5.3 Operační systém PlanetLabu

V této části jsou vysvětleny podrobnosti týkající se operačního systému PlanetLabu – softwaru nainstalovaném na každém uzlu, nad kterým je vystavěna abstrakce *slice*. Tento text rozvíjí obecnější informace prezentované v předchozích sekcích.

Operační systém PlanetLabu obsahuje:

- jádro Linuxu verze 2.4, které obsahuje:
  - patche pro *vservery*
  - hierarchický *token bucket* plánovač paketů
- SILK (Scout in Linux Kernel) modul [15], [4], který provádí plánování CPU, síťové účtování a bezpečné *raw* sockety
- manažera uzlu

*Slice* odpovídá distribuované množině virtuálních strojů. Každý virtuální stroj je implementován jako *vserver* [5]. Mechanismus *vserverů* je patch pro jádro Linuxu verze 2.4, který umožňuje na jednom fyzickém serveru provozovat více nezávislých virtuálních serverů. *Vservery* jsou přední mechanismy realizující virtualizaci v rámci jednoho uzlu. Navíc uvádí do kontextu jmenné prostory, např. identifikátory uživatelů a souborů.

*Vservery* poskytují omezená superuživatelská práva, díky nimž může *slice* spravovat svůj virtuální stroj jako by se jednalo o samostatný server. *Vservery* také spadají do kontejnerů zdrojů určených pro izolaci, tedy jsou jednotkou pro přidělování zdrojů.

Inicializace *vserveru* se skládá ze dvou částí perzistentního stavu – množinou SSH klíčů, která umožňuje přihlašování na *vserver* a souborem `rc.vinit`, který slouží jako skript pro bootování *vserveru*.

*Vservery* navzájem komunikují pomocí IP protokolu a ne pomocí místních socketů nebo jiných funkcí meziprocesové komunikace. Toto silné oddělení *sliců* zjednodušuje správu zdrojů a izolaci mezi *vservery*, poněvadž vzájemná interakce mezi *vservery* je nezávislá na jejich umístění. Jmenný prostor síťových adres (IP adres a čísel portů) ale není do kontextu zasazen, takže *slicy* na jednom uzlu sdílí IP adresy a čísla portů.

Každému *vserveru* na uzlu je přiřazen specifický *kontext zabezpečení* (security context) a každý proces je asociován s určitým *vserverem* skrz tento kontext. Kontext zabezpečení je přiřazován procesu přes nové systémové volání a je děděn následníky procesu. Izolace mezi *vservery* je dosažena rozhraním systémového volání použitím kombinace kontextu zabezpečení a UID/GID.

Pro zjednodušení vytváření a rušení *vserverů* a pro transparentní přesměrování uživatelů, kteří se přes SSH chtějí připojit na svůj *slice* do správného *vserveru*, bylo vytvořeno několik utilit. Inicializace *vserveru* začíná tak, že se v něm vytvoří obraz referenčního systému souborů. Dále se vytvoří dva shodné linuxové účty se stejným názvem, jako je název *slice*. Jeden v primárním *vserveru* uzlu a druhý v právě vytvářeném *vserveru*. Ty sdílí stejné UID. V primárním *vserveru* je defaultní shell nastaven na `/bin/vsh` – modifikovaný `bash` shell, který při přihlašování uživatele provádí tyto akce:

1. přepne na kontext zabezpečení *vserveru*, kterému odpovídá daný *slice*
2. provede *chroot* do souborového systému tohoto *vserveru*
3. vzdá se některých práv původního superuživatele (přístupu na *raw* zařízení)
4. přesměruje se na druhý účet do *vserveru*

Výsledkem je, že uživatelé, připojující se pomocí SSH/SCP, jsou transparentně přesměrováni do odpovídajícího *vserveru*, který tím pádem ani nepotřebuje vlastní SSH server.

Jak již bylo nastíněno v sekci 2.5.1, uzly PlanetLabu podporují dva speciální kontexty s přidanými možnostmi – *manažera uzlu* a *administrativní slice*. Kontext manažera uzlu běží se standardními právy superuživatele a zahrnuje:

- mechanismy pro vytváření *slice*, inicializaci jeho stavu a přiřazování zdrojů pro něj
- senzory pro zasílání informací o svém uzlu
- monitorování množství přenášených dat (traffic auditing service)

Administrativní *slice* poskytuje omezená privilegia pro správce, která mu umožňují spravovat uzel (bez poskytnutí plného superuživatelského přístupu) pomocí sady správních nástrojů. Správce pak má možnost např. nastavit maximální odchozí přenosovou rychlost na uzlu, ukončit libovolné procesy nebo spouštět `tcpdump` pro monitorování přenášených dat v lokální síti (viz obrázek 2.2).

Další informace týkající se operačního systému PlanetLab, především alokace zdrojů, izolace, virtualizace sítě a monitorování, jsou uvedeny v [14].

## 2.6 Přístup k PlanetLabu

K PlanetLabu mají přístup pouze členové registrovaných organizací. V České republice je v současné době u PlanteLabu registrováno pouze sdružení CESNET, skrz nějž k PlanetLabu přistupuje i VUT Brno.

Každý, kdo chce využívat PlanetLab, musí mít vytvořen účet. Toho lze dosáhnout vyplněním registračního formuláře na stránkách PlanetLabu, kde je nutné mimo jiné vybrat organizaci, skrz kterou se přistupuje a souhlasit s podmínkami přístupu definovanými v AUP (Acceptable Use Policy). Odeslaný formulář obdrží správce dané organizace a ten teprve může registraci potvrdit. S registrací také souvisí přiřazení *slice* k uživatelskému účtu. Každý uživatel může využívat pouze *slicy*, které má přiřazené od správce. O jeden *slice* se může dělit více uživatelů, v takovém případě spolu sdílí zdroje.

První věc, kterou musí nový uživatel udělat, je vytvořit si vlastní SSH pár soukromého a veřejného klíče, který pak bude používat pro autentizaci při připojování se k uzlům. Poté svůj veřejný klíč nahraje do databáze PlanetLabu.

Veškerý přístup ke zdrojům PlanetLabu se provádí skrz *slice*. Uživatel se může přihlásit pouze na uzly, které má jeho *slice* přiřazeny. Správu uzlů nad daným *slicem* lze provádět více způsoby. Buď přímo přes webové stránky PlanetLabu nebo pomocí PlanetLab API. Pokud uživatel do *slice* přidá nové uzly, znamená to, že se na daných uzlech vytvoří nové virtuální stroje pro daný *slice* (tato operace trvá několik minut, obvykle 10 – 15, takže na nové uzly není možné přihlásit se okamžitě).

Po přihlášení na uzel může každý uživatel získat administrátorská práva pomocí příkazu *su*. Účet *root* je implicitně bez hesla a má pouze malá omezení, která jsou nastavena kvůli bezpečnosti celého systému. Jako *root* může modifikovat kořenový systém souborů a má pravomoci instalovat software a modifikovat nainstalované balíčky.

## 2.7 PlanetLab API

PlanetLab Central API (PLCAPI) je rozhraní k centrální databázi PlanetLabu. Přes toto rozhraní by se k ní mělo přistupovat a měla by odtud být udržována. PLCAPI je využíváno webem, uzly, automatickými skripty, ale i uživateli pro přístup a aktualizaci informací o uživatelích, uzlech, lokacích, *slicech* a dalších entitách udržovaných databází.

### 2.7.1 Autentizace

K PLCAPI by se mělo přistupovat pomocí XML-RPC přes protokol HTTPS. Až na několik málo výjimek bere každé volání PLCAPI jako první argument autentizační strukturu. Všechny autentizační struktury vyžadují specifikovat `AuthMethod`. Jako autentizační struktury mohou být použity následující druhy autentizace:

- relace – je typicky validní 24 hodin. Relační klíč je možné získat voláním `GetSession` za použití jiné formy autentizace.

  ```
  AuthMethod session
  session    klíč relace
  ```

- heslem

```
AuthMethod password
Username   uživatelské jméno, typicky emailová adresa
AuthString autentizační řetězec, typicky heslo
```

- GnuPG – uživatelé mohou nahrát GPG veřejný klíč užitím `AddPersonKey`

```
AuthMethod gpg
name       uživatelské jméno, typicky emailová adresa
signature  GnuPG podpis v kanonizovaném tvaru XML-RPC zbytku argumentů
           volání
AuthString autentizační řetězec, typicky heslo
```

- anonymní

```
AuthMethod anonymous
```

### 2.7.2 Role

Funkce z PLCAPI mívají definovaná přístupová práva, tzn. které uživatelské role je mohou volat. Možné role jsou *admin*, *pi*, *user*, *tech*, *node* nebo *anonymous*. V některých případech funkce vrací různé výsledky – podle toho, která role ji volá.

*Node* a *anonymous* jsou pseudorole. Role *node* umožňuje volání automatizovanými skripty běžícími na uzlu, jako např. *Boot* či *Node Manager*. Funkce, které mohou být volány rolí *anonymous*, může volat kdokoli. Přesto musí být autentizační struktura specifikována.

### 2.7.3 Filtry

Filtr, jako jeden z argumentů většiny *Get* funkcí (tedy funkcí, které pouze získávají data), nám umožňuje vybrat pouze ty informace, které nás zajímají. Filtry mohou být identifikátory typu pole integerů, řetězce nebo struktury.

Filtry podporují některé speciality, např.:

- negaci výrazu ~
  `filter = { '~peer_id' : None }`
- operátory porovnání menší než `<`, menší nebo rovno `[`, podobně větší a větší nebo rovno
  ```
  filter = { ']node_id' : 2305 }    # node_id  >= 2305
  filter = { '>node_id' : 2305 }    # node_id  > 2305
  ```
- zástupné znaky pro vyhledávání (*) nebo (%)
  `filter = { 'hostname' : '*.cz' }`

### 2.7.4 PlanetLab shell

Pro usnadnění přístupu lze využít program PlanetLab shell (*plcsh*), který zjednodušuje práci s autentizační strukturou, a který je užitečný pro skriptování. Tento program je dostupný jako RPM balíček nazvaný PLCAPI a vyžaduje nainstalovaný Python (min. verze 2.4).

Jakmile se spustí program se zadanými parametry – URL k API a uživatelské jméno, není dále třeba při volání API funkcí zadávat autentizační strukturu. Poněvadž je tento

program vlastně interpret Pythonu, lze vytvářet proměnné, vykonávat cykly, importovat jiné balíčky atd. přímo z příkazové řádky stejně jako v regulerním shellu Pythonu. Pokud by bylo třeba přistupovat k *plcsh* programově, stačí naimportovat *PLC.Shell* modul.

Příklad použítí *plcsh* lze demonstrovat např. na funkci `GetNodes`, se zadanými parametry id uzlu a filtr jako id a název uzlu:

```
[user@site.edu]>>> GetNodes([121], ['node_id', 'hostname'])
[{'node_id': 121, 'hostname': 'planetlab-1.cs.princeton.edu'}]
```

### 2.7.5 Rozhraní

PLCAPI se v současné době dělí do tří rozdílných rozhraní:

- Registry interface – nastavuje a zjišťuje informace o objektech
- Management interface – konfiguruje/rebootuje komponenty a zjišťuje jejich stav
- Slice interface – zaopatřuje a řídí *slicy/slivery*

Kompletní dokumentace k PlanetLab API je dostupná v [7].

## 2.8 Existující nástroje

Pro PlanetLab, zejména pro správu *slice*, bylo uživatelskou komunitou vyvinuto několik nástrojů, které zpřehledňují a zjednodušují správu *slice*. Mezi zmiňované patří např. pShell, pssh, Stork nebo AppManager.

### 2.8.1 pShell

Tento nástroj [8], vyvinutý na McGill University, poskytuje rozhraní podobné linuxovému shellu, v němž je možné zadávat příkazy, které manipulují s cílovým *slicem*. Funguje tak jako správní centrum, které běží na lokálním počítači.

Aplikace je napsána v Pythonu a k celé řadě příkazů využívá PlanetLab API. Mezi nejvýznamnější příkazy patří:

- **plist** – výpis uzlů PlanetLabu a jejich stavu
- **addnodes** – přiřazení nového uzlu ke *slici*
- **delnodes** – odebrání uzlu ze *slice*
- **slist** – výpís uzlů přiřazených ke *slici*
- **srenew, expdate** – obnoví dobu platnosti *slice*
- **chkstat** – ověří stav uzlů přiřazených ke *slici*
- **install** – instalace RPM balíčků
- **put** – uploaduje soubory z lokálního počítače na uzel
- **get** – stánhe soubory z uzlu na lokální počítač
- **cmd** – spustí příkaz na uzlu

### 2.8.2 pssh

Celkem užitečným nástrojem je *pssh* [9], pokud chceme spravovat větší množství serverů současně. Jedná se o program, který poskytuje paralelní verze *openssh* nástrojů. Mezi ně patří:

- **pssh** – paralelní *ssh* (zabezpečené připojení na systém)
- **pscp** – paralelní *scp* (zabezpečený přenos souborů)
- **prsync** – paralelní *rsync* (synchronizace souborů a adresářů)
- **pnuke** – paralelní *nuke* (okamžité ukončování procesů)
- **pslurp** – paralelní *slurp* (vzdálené kopírování souborů)

### 2.8.3 Stork

Stork [10] je instalační utilita podobná známějším utilitám jako např. YUM nebo APT. Najde uplatnění nejen v PlanetLabu, ale i pro běžné použití. Podporuje centrální správu updatů balíčků a také nový bezpečnostní systém, který posiluje roli uživatelů a tvůrců balíčků, přičemž snižuje potřebnou důvěru ve správce úložiště balíčků. Stork představuje bezpečný, škálovatelný a efektivní způsob správy tarballů a RPM balíčků v rozsáhlých sítích.

Pro snadnější a rychlejší použití je také možné využít grafickou nadstavbu – Stork Slice Manager.

Rychlost stahování je zvýšena využitím systémů BitTorrent, Coral, FTP, CoBlitz, CoDeeN a HTTP. Bezpečnost balíčků je zajištěna asymetrickým šifrováním. Na balíčky vedou bezpečné reference a instalace balíčků je řízena konfiguračním souborem, který dovoluje uživatelům přesně specifikovat, jaký software může běžet ve *slici*.

### 2.8.4 AppManager

Za účelem snadnějšího používání PlanetLabu byl vyvinut Application Manager [6], který pomáhá s nasazením, monitorováním a spouštěním aplikací v PlanetLabu. Umožňuje centrálně spravovat, instalovat, upgradovat, spouštět a ukončovat aplikace. Jeho cílem je ale spíše monitorování a kontrola instalovaných aplikací.

# Kapitola 3

# Požadavky na systém

Za normálních okolností se člověk, který chce získat přístup do PlanetLabu, zaregistruje na webu, aby získal svůj vlastní účet a mohl se do systému přihlašovat pomocí vlastního privátního klíče. Většinou k tomu získá i svůj vlastní *slice*. Avšak ne vždy nám tato koncepce vyhovuje.

V případě, že chceme umožnit přístup do PlanetLabu třeba skupině studentů nějakého kurzu, potřebujeme flexibilnější metodu vytváření a rušení účtů, než klasickým způsobem.

Navíc, pro vytvoření nového uživatelského účtu je zapotřebí souhlas od správce, což může být zbytečná komplikace, pokud např. univerzita žádného správce nemá (sama není přímo registrovaná v PlanetLabu) a musela by s ním komunikovat externě. V tomto případě by bylo vhodnější, kdyby účty mohl spravovat vedoucí kurzu sám.

Vzniká zde tedy myšlenka vytvářet virtuální účty, které by byly vystavěny nad jedním existujícím účtem. Tím pádem tedy i nad jedním *slicem*, poněvadž každá registrovaná organizace má k dispozici pouze omezený počet *sliců* (typicky 10), což nemůže pokrýt větší množství uživatelů, a navíc by to bylo plýtvání zdrojů, pokud by pro jeden kurz mělo být alokováno větší množství *sliců*.

Takovéto rozdělování *slice* na další díly není nijak podporováno v současném PlanetLabu. *Slice* zde totiž slučuje dvě různé abstrakce:

- odděluje od sebe různé uživatele
- je jednotkou pro přidělování zdrojů

Cílem tohoto projektu je vytvořit systém, který bude implementovat správu virtuálních účtů pro přístup k PlanetLabu pro více uživatelů, s využitím jednoho běžného uživatelského účtu.

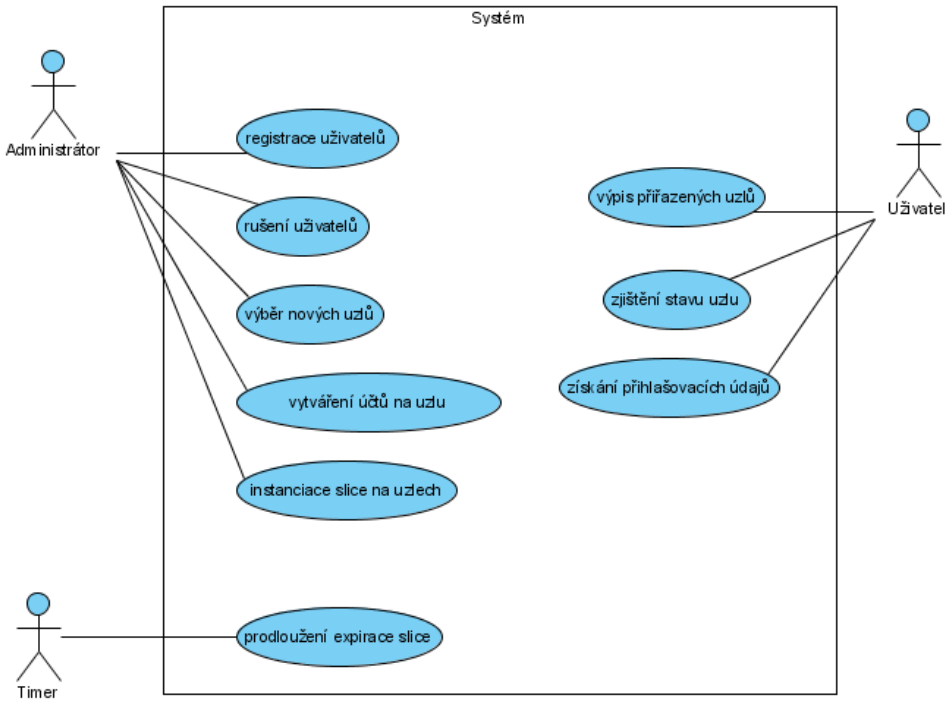
## 3.1 Specifikace požadavků

Virtuální účet zde budeme nazývat takový účet, díky němuž se bude moci náš uživatel připojovat na vybrané uzly, přičemž vytváření a rušení těchto účtů bude řízeno implementovaným systémem. Jedná se pak o jakousi abstrakci přihlašovacího účtu z pohledu PlanetLabu.

Systém bude především určen pro vedoucí kurzů distribuovaných systémů. Měl by umožňovat zejména těmto vedoucím spravovat přihlašovací účty pro studenty a těmto studentům pak poskytovat bezpečnou cestou přihlašovací údaje, díky nimž se budou moci sami

připojit na určené uzly. Každému studentovi bude možné přiřadit jinou podmnožinu uzlů ze všech, na kterých je daný *slice* instanciován. Dále by systém měl umožňovat provádět základní správní operace přímo s daným *slicem* – instanciovat *slice* na vybraných uzlech, na kterých ještě instanciovaný není, a prodlužovat expirační dobu *slice*. Tyto operace je sice v současnosti možné provádět přímo ve správě účtu na stránkách PlanetLabu, ale jejich zahrnutí do implementovaného systému sjednotí a zjednoduší správu.

Na obrázku 3.1 je znázorněn diagram případů použití pro implementovaný systém. Činitel *Timer* zde představuje roli, která v systému reprezentuje plánovač úloh nebo časovač, který dokáže po určitém časovém intervalu opakovaně provádět zadané úlohy.




Obrázek 3.1: Diagram případů použití systému

Typický scénář vytvoření virtuálního uživatelského účtu by mohl vypadat takto:

1. Administrátor se přihlásí do systému
2. Zvolí uživatele, kterým chce přiřadit uzly
3. Z nabídky vybere uzly, které jim chce přiřadit
4. Systém na přiřazených uzlech vytvoří účty, přes které se na ně bude možné přihlásit
5. Uživatel se přihlásí do systému, zjistí si přihlašovací údaje o uzlech, které mu byly přiřazeny, a na ty se bude moci dále připojit

## 3.2 Bezpečnost

Aby mohli uživatelé využívat systém vzdáleně, je třeba, aby byl dostupný online na síti či na Internetu. Z tohoto důvodu bude třeba zajistit určité bezpečnostní mechanismy.

Síťová bezpečnost se týká třech hlavních principů - důvěrnosti (confidentiality), integrity a dostupnosti (availability). Záleží pak na aplikaci a na kontextu, který z těchto principů bude klíčový pro zajištění bezpečnosti.

**Důvěrnost:** Týká se zabránění přístupu k citlivým informacím. Prozrazení může být úmyslné – např. prolomení šifry k získání informací, nebo neúmyslné – dané tím, že osoba manipulující s daty je nepozorná nebo nedbalá.

**Integrita:** Skládá se ze tří cílů:

- Prevence před modifikací informace neautorizovanou osobou
- Prevence neúmyslné modifikace informací autorizovanými uživateli
- Zajištění konzistence – musí si odpovídat skutečné informace s informacemi uloženými např. v databázi.

**Dostupnost:** Zajišťuje, že autorizovaní uživatelé budou mít přístup k informacím v systému kdykoliv budou potřebovat.

Další důležité pojmy, spojované s předcházející trojicí, jsou:

- Identifikace – poskytnutí identity systému jako např. login
- Autentizace – ověření poskytnuté identity např. skrz heslo
- Účtovatelnost – stanovení akcí a chování jedné individuální osoby v systému a udržování zodpovědnosti za její akce
- Autorizace – nastavená oprávnění pro osobu, které je umožněn přístup do systému

Pro účely implementovaného systému bude třeba klást důraz na zajištění důvěrnosti a integrity dat. Před použitím systému bude vyžadována autentizace každého uživatele.

# Kapitola 4

# Analýza a návrh řešení

Přihlášení se na uzel na daný *slice* je v PlanetLabu možné pouze přes SSH za použití privátního klíče patřícího registrovanému uživateli PlanetLabu, jehož veřejný klíč je uložen v centrální databázi. Přihlašovacím jménem je pak název *slice*, ke kterému registrovaný účet náleží. Po takovémto přihlášení je člověk pod tímto uživatelským jménem přihlášen. Jak však umožnit přístup (a kontrolovat ho) na daný *slice* i ostatním námi vybraným uživatelům, kteří v PlanetLabu registrovaní nejsou, s využitím tohoto jednoho skutečného přihlašovacího účtu? Prostou možnost nahrát všem těmto osobám soukromý RSA klíč registrovaného uživatele pro přímé přihlášení neuvažujeme, protože by zde nebyla možnost kontrolovat a řídit jejich přístup, což je jedním z hlavních požadavků.

## 4.1 Možnosti řešení

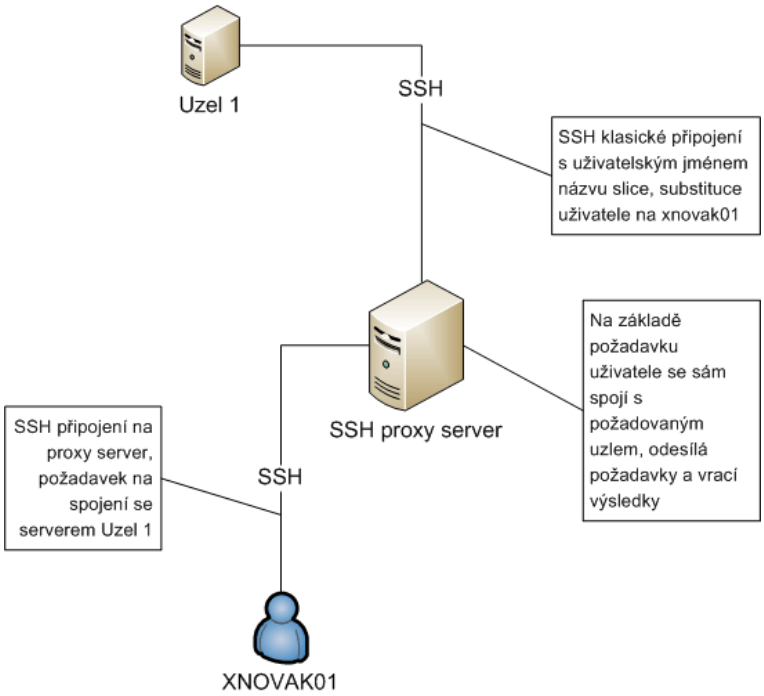
Je několik možných přístupů, jak problematiku virtuálních účtů řešit. Všechny přístupy však mohou k realizaci virtuálních účtů využít možnosti vytváření klasických uživatelských účtů na uzlech. Každému virtuálnímu uživateli se tedy vytvoří systémový účet na požadovaném uzlu spolu s domovským adresářem, ke kterému bude mít přístup kromě administrátora pouze tento uživatel. Tímto způsobem se vlastně oddělí různé uživatelské prostory a na uzlu bude moci pracovat více uživatelů, aniž by se vzájemně ovlivňovali. V rámci uzlu se tedy virtualizovaný účet bude rovnat klasickému systémovému účtu.

Řešeným problémem nyní je, jakým způsobem se virtuální uživatel bude moci na daný systém vzdáleně přihlásit. Je třeba implementovat mechanismus, prostřednictvím něhož to bude možné. Nabízejí se dvě hlavní možná řešení – implementace „SSH proxy serveru“ nebo instalace dedikovaných SSH serverů.

### 4.1.1 SSH proxy server

Jednou z možností vzdáleného přístupu je vytvořit jakýsi „SSH proxy server“. Pokud by se virtuální uživatel chtěl připojit na některý jemu přiřazený uzel, připojil by se přes SSH na tento proxy server, který by se na daný uzel sám připojil klasicky pomocí registrovaného účtu. Na cílovém uzlu by se pak přihlášený uživatel substituoval za uživatele, který připojení požaduje. Takto by se vlastně tunelovalo spojení virtuálního uživatele přes existujícího uživatele. Proxy server by zasílal příkazy uzlu a výstup by vracel uživateli. Jednoduché schéma takovéhoto systému je znázorněno na obrázku 4.1.

Komunikace s uzlem by tedy probíhala takto:




Obrázek 4.1: Připojení uživatele na uzel přes SSH proxy server

1. uživatel se připojí na proxy server s požadavkem spojení na zvolený uzel
2. proxy server ověří, zdali uživatel má práva přihlašovat se na daný uzel
3. proxy server se připojí na daný uzel a substituuje přihlášené uživatelské jméno za jméno požadovaného uživatele
4. uživatel zadá příkaz, proxy server jej přepošle na uzel
5. uzel příkaz vykoná a vrátí proxy serveru výsledek příkazu
6. proxy server vrátí výsledek příkazu uživateli
7. dále se pokračuje bodem 4, nebo uživatel spojení ukončí

Výhodou tohoto řešení by bylo například to, že na uzly by se nemuselo nic nového instalovat a přístup by byl pro PlanetLab víceméně transparentní.

Nevýhodou by bylo to, že daný proxy server by byl jedním slabým místem celého systému, poněvadž uživatelé by se na všechny uzly připojovali právě přes něj. Při jeho výpadku by nebylo možné se vůbec na uzly připojit.

### 4.1.2 Dedikované SSH servery

Druhá možnost vzdáleného přístupu virtuálních uživatelů na uzly spočívá v instalaci SSH serveru zvlášť na každý využívaný uzel. Uživatelé by se tedy připojovali přes SSH přímo na daný uzel.

Správu dedikovaných SSH serverů by měl na starosti řídicí server, který by na základě požadavků administrátora instaloval SSH servery na vybrané uzly a udržoval je ve funkčním

stavu. Dále by na uzlech vytvářel a udržoval uživatelské účty uživatelů. Tento řídicí server by se na uzly připojoval klasickým způbem, tedy přes SSH na portu TCP\22, avšak dedikované SSH servery by musely být spouštěny na jiných portech, právě protože klasický vyhrazený port je již obsazen.

K výhodám tohoto přístupu patří zejména to, že by se uživatelé mohli připojovat přímo na dané uzly. Složitost systému bude také menší, protože implementovaný systém pak bude realizovat pouze správní část a část, ve které se budou požadovaná nastavení na uzlech vykonávat.

Nevýhodou tohoto řešení je, že by bylo třeba spravovat velké množství vzdálených SSH serverů, zajistit jejich bezpečnost a udržovat je v provozu.

## 4.2 Struktura systému

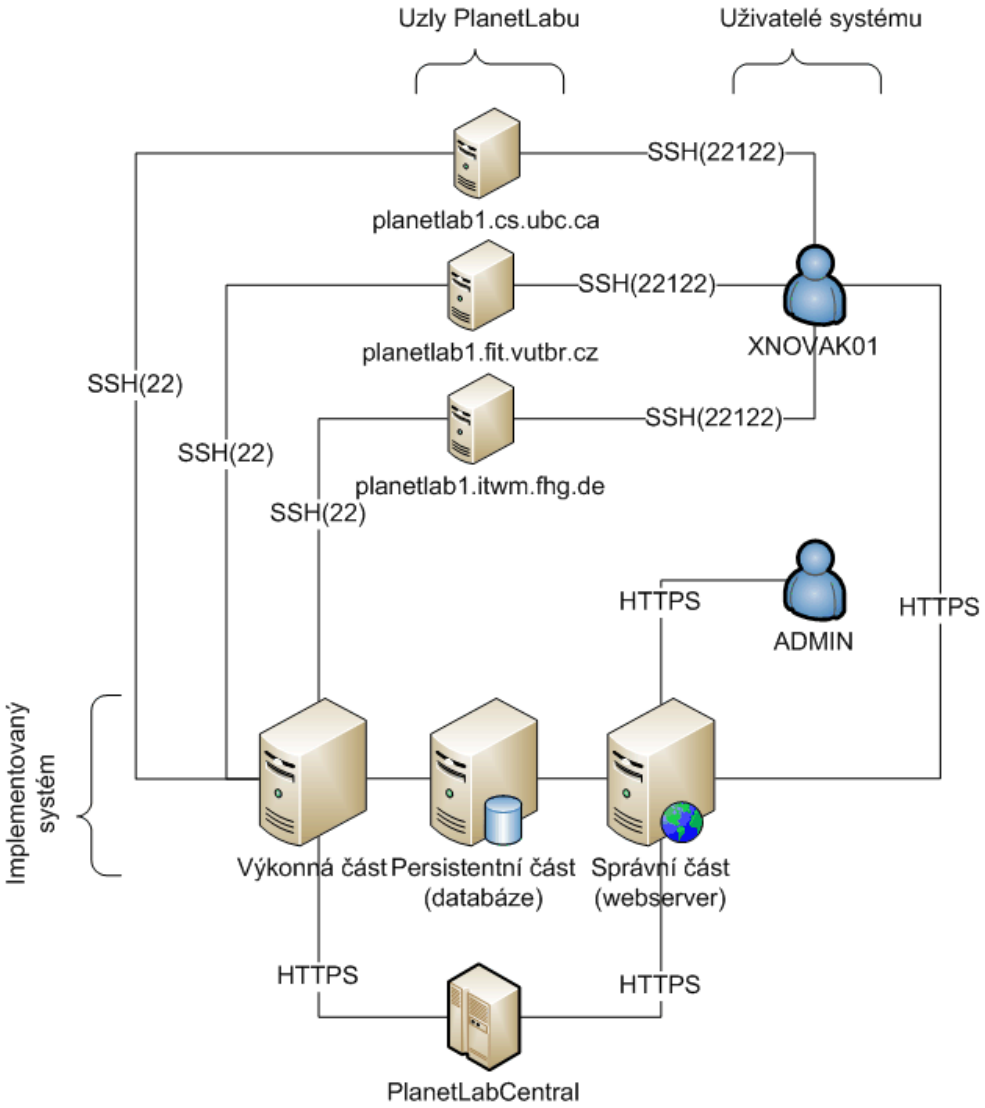
Řešení s dedikovanými SSH servery se zdá být pro daný účel vhodnější a použitelnější než řešení s SSH proxy serverem. Detailnější návrh a následná implementace systému se tedy bude týkat právě tohoto řešení.

Vzhledem k zadaným požadavkům bude systém složen z více různých částí, kde každá z nich bude realizovat jistou specifickou část funkčnosti celku. Systém se bude skládat z části správní, z části persistentní, uchovávající nastavení systému, a z části výkonné, tedy části, která bude fyzicky realizovat požadavky.

Schéma vzájemné komunikace systému, uživatelů a uzlů PlanetLabu je znázorněno na obrázku 4.2. Jsou tam vyobrazeny dvě uživatelské role. Osoba *XNOVAK01* reprezentuje jednoho z mnoha uživatelů implementovaného systému (virtuální účet), který má možnost připojit se jednak do správní části (přes HTTPS), kde zjistí, na které uzly se může připojit a jak, a jednak na uzly dle získaných údajů – zde znázorněn příklad připojení na 3 uzly PlanetLabu pomocí SSH na port 22122, na kterém naslouchají nainstalované dedikované SSH servery. Druhou zobrazenou osobou je *ADMIN* – tedy administrátor implementovaného systému, který po připojení do správní části systém konfiguruje. Potřebná nastavení na uzlech PlanetLabu pak provádí výkonná část systému, která se připojuje na uzly klasickým přihlašovacím mechanismem pro běžné uživatele PlanetLabu – tzn. má k dispozici přihlašovací údaje jednoho běžného uživatelského účtu a připojuje se na uzly pomocí SSH na port 22.

Je třeba však připomenout, že připojování na uzel zde znamená připojení na server do daného *slice*. Kromě implementovaného systému mají na uzel přístup i všichni ostatní uživatelé PlanetLabu, kteří jsou registrovaní ke stejnému *slici*, jako využívá systém, protože implementovaný systém se připojuje na uzel přes běžný uživatelský účet. Na obrázku 4.3 je znázorněna situace, jak je na uzel připojen běžný uživatel, implementovaný systém (také vlastně běžný uživatel) a virtuální uživatel *XNOVAK01*. Všichni se připojují na stejný *slice*, avšak běžný uživatel využívá běžného připojení, virtuální uživatel využívá dedikovaného SSH serveru. Virtuální uživatel ani nemusí mít žádné ponětí o tom, co je to *slice* ani na který se připojuje. Automaticky se totiž připojí na *slice*, na kterém běží dedikovaný SSH server.

Ideálním případem pro implementovaný systém by bylo, kdyby využíval *slice*, ke kterému by byl registrovaný pouze jeden uživatelský účet, který by byl vyhrazen pouze pro tento implementovaný systém. Nemohla by pak nastat situace, že jiný registrovaný uživatel by, třeba i nevědomě, narušil systém.



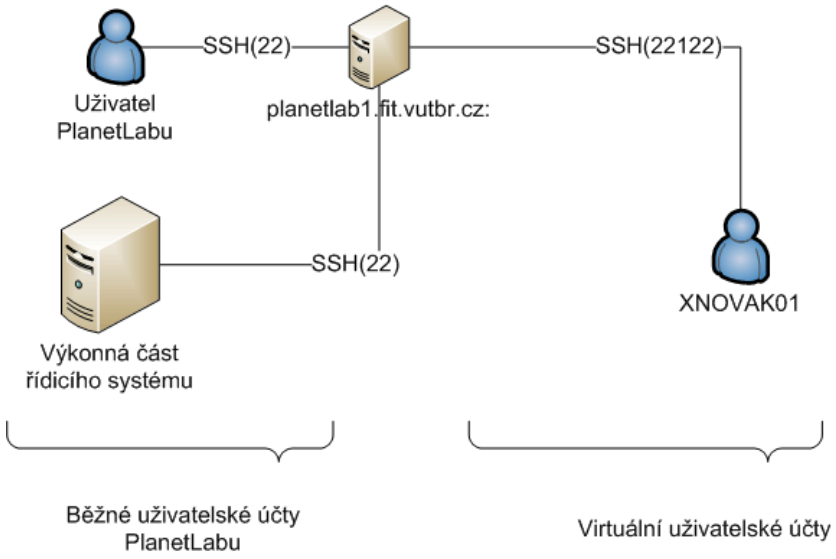

Obrázek 4.2: Schéma vzájemné komunikace systému, uživatelů a uzlů PlanetLabu

### 4.2.1 Správní část

Správa systému bude realizována formou webových stránek. Hlavní výhoda použití webu spočívá v nezávislosti na používané platformě pro přístup k systému a také v přístupu odkudkoliv z Internetu.

Webové rozhraní bude rozděleno na část administrátorskou a uživatelskou. Administrátor bude moci:

- vybírat servery z celého PlanetLabu, které bude chtít využít pro virtualizovaný přístup
- spravovat vybrané uzly, tzn.:
  - zjišťovat stav
  - spouštět SSH server




Obrázek 4.3: Běžný uživatel a virtuální uživatel na stejném uzlu a *slici*

  - vypínat SSH server
  - přiřazovat je do uživatelských skupin
  - odebírat z uživatelských skupin

- spravovat uživatelské účty pro přístup do systému a tím i pro přístup na virtualizované účty

V uživatelské části bude možné:

- získávat informace o uzlech přiřazených uživateli, konkrétně:
  - název uzlu
  - port, na který se mohou připojit přes SSH
  - příhlašovací údaje
  - současný stavu uzlu

V zájmu bezpečnosti bude vyžadován šifrovaný přístup na stránky přes protokol HTTPS.

### 4.2.2 Persistentní část

Roli úložiště informací, nastavení a stavů bude vykonávat databáze. Ta bude jakýmsi prostředníkem mezi správní a výkonnou částí. Ze správního systému sem budou ukládána veškerá data a požadovaná nastavení. Databáze pak bude řídicím mechanismem pro výkonnou část, která bude získané informace o nastavení systému fyzicky realizovat na vzdálených uzlech. Informace o reálném stavu uzlů budou pak zpětně do databáze reflektovány včetně chybových logů.

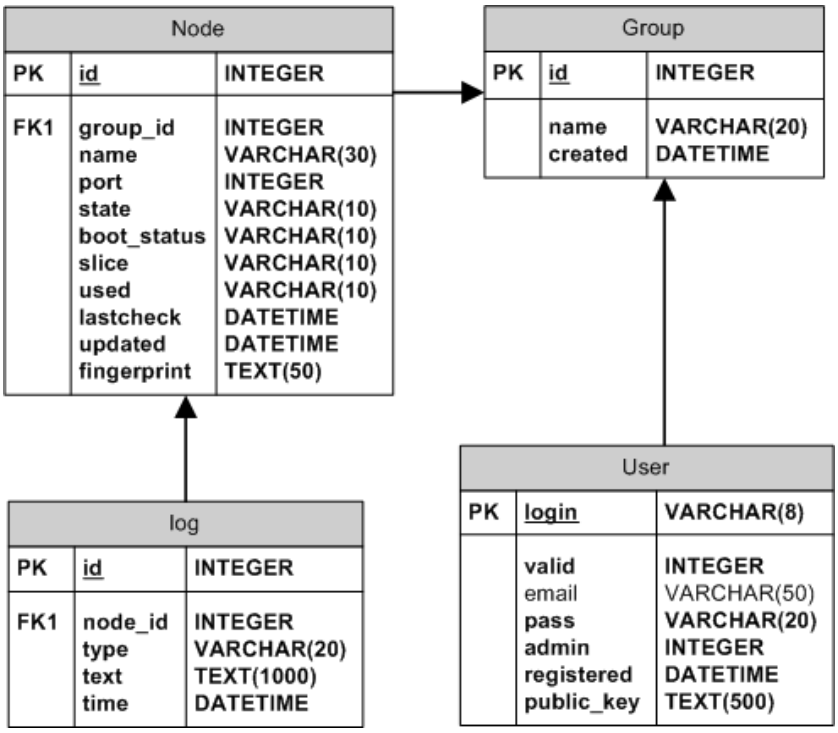
Návrh schématu databáze je znázorněn na obrázku 4.4. Účel většiny položek ve schématu je zřejmý z jejich názvu. V databázi budou čtyři tabulky.

1. **Node** – Reprezentuje spravovaný uzel. Obsahuje zejména informace o svém SSH serveru – stav, v jakém se nachází, port, na kterém je spuštěn, otisk (fingerprint) veřejného klíče a časy, kdy došlo ke změnám.

2. **User** – Uživatel systému – je identifikován jednoznačným loginem, dále obsahuje příznak platnosti (valid), položku admin, která určuje, zdali se jedná o správce, informace o datu vytvoření účtu a veřejný klíč, jenž spolu s komplementárním soukromým klíčem bude využívat k autentizaci na uzly.

3. **Group** – Skupina uživatelů a uzlů – určuje, který uživatel má přístup na který uzel.

4. **log** – Informace o nečekaných chybách či provedených úkonech. Je spjata s jedním konkrétním uzlem. Položka typ zde identifikuje zdroj chyby.




Obrázek 4.4: Logické schéma databáze

### 4.2.3 Výkonná část

Stěžejní částí celého systému bude právě výkonná část, která bude mít na starosti komunikaci se vzdálenými uzly, provádění požadavků na nich a zajištění toho, že reálný stav na uzlech bude odpovídat požadovanému stavu uloženému v databázi.

Tato část bude implementována sadou skriptů, které budou periodicky volány v prostředí řídicího serveru. Na uzly se budou připojovat pomocí SSH skrz registrovaný účet PlaneLabu, patřící k odpovídajícímu *slici*.

Skripty budou mít dvě stěžejní úlohy:

1. nainstalovat SSH server na každý nově registrovaný uzel v systému, nakonfigurovat ho, spustit a poté udržovat jeho stav dle požadavků
2. vytvářet požadované uživatelské účty na každém uzlu, popřípadě je zase odebírat

Ohledně správy SSH serverů bude třeba vytvořit skripty, které budou testovat, zdali se SSH server skutečně nachází v požadovaném stavu. Pokud má běžet, testovat jestli běží, pokud má být zastaven, testovat jestli je opravdu zastaven. Pokud by si stavy v databázi a ve skutečnosti vzájemně neodpovídaly, je nutné nekonzistenci napravit a uvést SSH server fyzicky do požadovaného stavu. Pro případ výskytu chyby bude existovat skript, který se ji bude snažit odstanit. Podobně uživatelské účty na uzlech budou opakovaně kontrolovány a v případě změny požadavků dané účty odstraní nebo nové účty vytvoří.

## 4.3 Správa skupin uživatelů a serverů

Přiřazovat ručně ke každému uživateli server, nebo ke každému serveru uživatele, by bylo zbytečně zdlouhavé a navíc nepřehledné. Každý uživatel by mohl mít libovolné množství přiřazených serverů, mohli by je libovolně sdílet a situace by tím byla velmi nepřehledná. Proto je vhodné specifikovat nějaká pravidla, kdo může využívat který server.

Toho lze dosáhnout tím, že uživatelé a servery budou rozděleni do skupin, které budou vzájemně disjunktní. Každý server může být pouze v jedné skupině a stejně tak i každý uživatel může patřit pouze do jedné skupiny. Tím pádem uživatel bude mít přiřazenou množinu serverů tím, že bude patřit do určité skupiny, a server bude obsahovat pouze ty uživate, kteří budou přiřazeni do stejné skupiny jako on. Pokud server bude odebrán ze skupiny, pak z něj budou odebráni všichni uživatelé a bude moci být přiřazen do jiné skupiny. Příklad rozdělení na skupiny demonstruje obrázek 4.5.

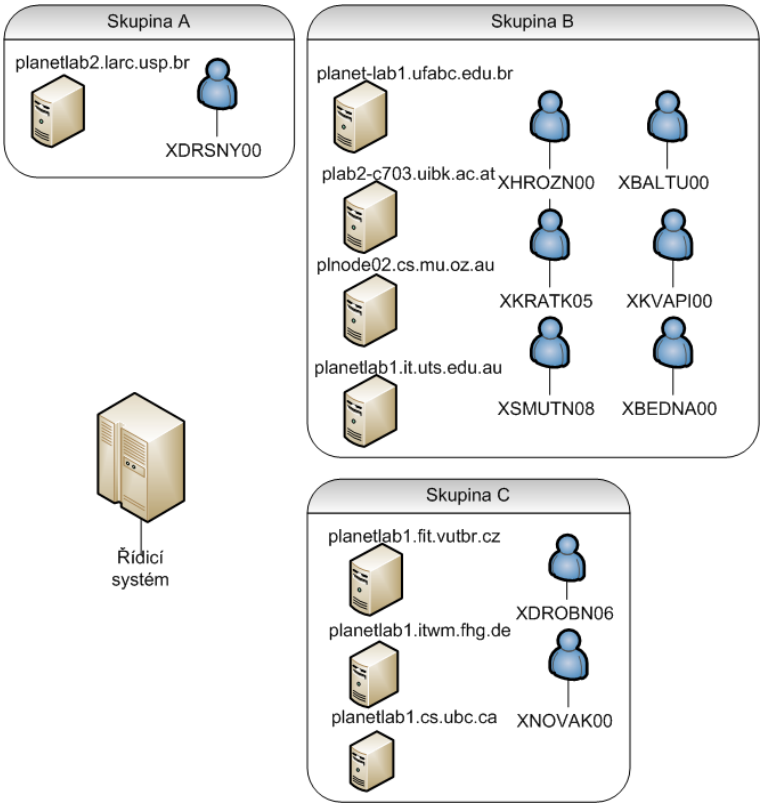
## 4.4 Centralizované řízení

Pokud má být spravována větší skupina uzlů, je možnost rozhodnout se pro dvě různé strategie řízení:

- centralizované
- decentralizované

V případě decentralizovaného řízení by byly řídicí skripty vykonávány přímo na spravovaných uzlech. Kontrola funkčnosti serveru by tedy probíhala lokálně a navíc by nezdržovala centrální řídicí server. V případě chyby by byl kontaktován centrální server a byla by mu nahlášena chyba. Problém by zde byl ale v komunikaci mezi centrálním serverem a delegovanými skripty. Celá vzájemná komunikační infrastruktura by byla složitější a problematická.

V případě centralizovaného řízení je řízení celé skupiny uzlů na jednom místě. Veškerá komunikace s uzly ale probíhá vzdáleně, tedy přes SSH, tzn. každé zjištění stavu, nastavení a instalace probíhá vzdáleně a musí být kvůli tomu otevřeno nové spojení. I přesto se tato strategie zdá být vhodnější a použitelnější k danému účelu.




Obrázek 4.5: Příklad rozdělení do skupin

## 4.5 Stavy uzlů a jejich SSH serverů

Správa uzlů je založena na množině stavů, ve kterých se mohou uzly nacházet. Stav uzlu je odvozen od fyzického stavu jeho SSH serveru. Stavy se dělí do dvou skupin – stavy *trvalé* a *přechodné*. Trvalý stav je takový, který je požadován jako cílový, přechodný stav je pak takový, který je nastaven za účelem fyzického provedení změny a trvá pouze po dobu od požadavku do provedení změny. Tedy např. pokud chce administrátor zastavit SSH server na uzlu, nastaví mu přechodný stav, který vyjadřuje požadavek na zastavení. Tento přechodný stav je signálem pro výkonnou část systému, aby požadavek provedla. Dokud není požadavek splněn, je uzel stále v tomto přechodném stavu. Jakmile výkonná část požadavek splní, teprv nastaví trvalý stav, vyjadřující úspěch provedené změny. V uvedeném příkladu by tedy výkonná část fyzicky SSH server zastavila a poté by nastavila trvalý stav, který by vyjadřoval, že byl zastaven.

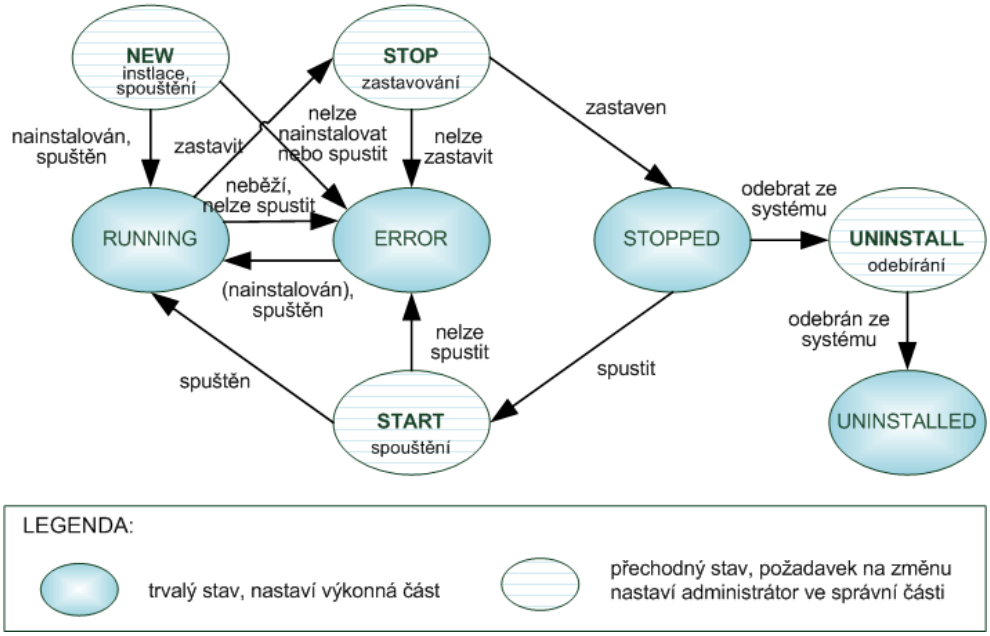
Přechodný stav nastavuje správní část systému na základě požadavku administrátora, trvalý stav nastavuje výkonná část systému po provedení změny. Konkrétní navržené stavy jsou tyto:

- přechodné
  - **NEW** – uzel byl právě přidán mezi spravované uzly. Je výchozím pro každý nově přidaný uzel. Provede se instalace SSH serveru a poté se spustí.
  - **START** – požadavek na spuštění zastaveného SSH serveru

  - **STOP** – požadavek pro zastavení SSH serveru, který je spuštěný
  - **UNINSTALL** – požadavek na odinstalaci SSH serveru a odebrání uzlu ze systému

- trvalé
  - **RUNNING** – byl spuštěn SSH server
  - **STOPPED** – byl zastaven SSH server
  - **UNINSTALLED** – SSH server byl odinstalován a uzel odebrán ze systému
  - **ERROR** – chybový stav, SSH server nelze nainstalovat, spustit nebo zastavit

Stav ERROR je trochu odlišný od ostatních stavů. Je zařazen mezi trvalé stavy, přestože není požadován jako cílový. Poněvadž je však nastavován výkonnou částí, je zařazen právě jako trvalý. Na obrázku 4.6 jsou všechny tyto stavy zobrazeny spolu s jejich možnými přechody a akcemi (u přechodných stavů), které se při nich provádějí.




Obrázek 4.6: Stavy SSH serveru

Počátečním stavem je stav NEW, tzn. bude nastaven, když bude nový uzel přidán mezi spravované uzly. Poté na něj bude nainstalován SSH server a ten bude spuštěn – stav se změní na RUNNING. V tuto chvíli je možné se na server připojovat z uživatelských účtů. Pokud bude správce chtít server vypnout, nastaví mu stav STOP. Poté řídicí skripty SSH server vypnou a přejde se do stavu STOPPED. Pokud administrátor bude chtít server opět spustit, nastaví mu stav START. Řídicí skripty pak server opět spustí a nastaví stav RUNNING. Pokud v některém kroku nastane neočekávaná chyba, bude serveru nastaven stav ERROR. Jakmile se problém vyřeší, SSH server bude spuštěn a přejde do stavu RUNNING.

Administrátor má také možnost odebrat uzel ze systému. Než tak může učinit, je nutné nejprve odebrat tento uzel z uživatelské skupiny, aby z něj byly odstraněny uživatelské účty,

a vypnout SSH server. Jakmile se tak stane (stav STOPPED), administrátor může zadat požadavek na odstranění – tím se přejde do stavu UNINSTALL, systém odinstaluje SSH server a přejde se do stavu UNINSTALLED. V tu chvíli je uzel odebrán ze systému. Později je možné jej opět do systému zahrnout.

## 4.6 Bezpečnost systému

Uživatelům systému je třeba zajistit takovou míru bezpečnosti, aby jejich účty nemohly být zneužity neoprávněnou osobou. Uživatelé budou přistupovat do správního systému a na samotné uzly, takže bezpečnost přístupu je nutné zajistit na obou těchto místech.

### 4.6.1 Přístup do správní části

Přístup do správní části systému musí být šifrovaný a bude autentizován pomocí uživatelova hesla.

Registraci nového účtu do systému bude provádět každý uživatel sám s tím, že jeho platnost bude potvrzovat správce systému. Dokud nebude účet potvrzen, nebude možné se do systému přihlásit. Při registraci si uživatel sám zvolí heslo, které bude pro přístup používat. To je jedna z hlavních výhod osobních registrací. Heslo se totiž nemusí nikde přenášet, jak by tomu bylo, pokud by uživatelské účty vytvářel správce sám a generoval k nim hesla. Uživatel by si však měl registrovat bezpečné heslo, které nebude lehce uhádnutelné nebo rozlomitelné slovníkovým útokem.

### 4.6.2 Přístup na uzel

Zásadní otázkou přístupu přes SSH je typ zvolené autentizace. Mezi dva nejběžnější způsoby patří autentizace heslem nebo pomocí asymetrické kryptografie. Oba dva způsoby mohou být aktivní současně.

Autentizace heslem je nejběžnější způsob autentizace. Pro přihlášení je třeba mít na cílovém počítači vytvořený uživatelský účet, který má nastavené heslo. Pokud se uživatel připojí, je po něm požadováno zadání loginu a hesla.

Cílový počítač tak ověřuje identitu uživatele pomocí znalosti tajemství. Zde hrozí nebezpečí, že pokud by uživatel použil slabé heslo nebo heslo, které je vyzrazené, ohrozil by tím bezpečnost celého systému.

Druhou možností je autentizace pomocí asymetrické kryptografie. V tom případě potřebuje uživatel vlastnit soukromý a veřejný klíč. Soukromý klíč je jeho tajemství – k němu nesmí umožnit přístup žádnému jinému uživateli. Ten se uplatní při ověřování identity, kdy je jeho vlastník jako jedinný schopen dešifrovat zprávu zašifrovanou jeho veřejným klíčem. Proto uživatelův veřejný klíč vždy potřebuje subjekt, který ověřuje jeho identitu.

Autentizace pomocí asymetrických klíčů je považována za bezpečnější, poněvadž nikdo jiný než vlastník soukromého klíče se místo něj nemůže podvodně autentizovat. Proto bude přístup na uzly vyžadovat autentizaci právě pomocí asymetrické kryptografie. Pro dané účely bude vhodné použít protokol RSA, který tuto metodu implementuje.

# Kapitola 5

# Implementace

Vyvrcholením celé této práce je funkční systém, který je implementován dle předchozího návrhu. V této kapitole jsou shrnuty nejdůležitější informace týkající se implementace – detailnější popis využitých principů a technologií.

## 5.1 Využití PlanetLab API v systému

Jak již bylo zmíněno v sekci 2.7, PlanetLab API je rozhraní, skrz které je možné přistupovat k centrální databázi PlanetLabu a tím nastavovat různé parametry. V implementovaném systému jsou využity některé funkce z API, které umožňují spravovat *slice* nebo vypisovat informace o uzlech. Jsou jimi:

- GetNodes
- SliceNodesList
- UpdateSlice
- AddSliceToNodes

Všechny tyto funkce je oprávněn provádět i *user* – tedy role, která je pro nás k dispozici jakožto uživatel PlanetLabu. Tato role má pouze omezená privilegia a může provádět pouze minimum funkcí, které něco nastavují.

### 5.1.1 GetNodes

Tato funkce je využita pro výpis všech uzlů v PlanetLabu a jejich dalších informací. Pro nás jsou nejdůležitější informace název uzlu, id uzlu a stav, ve kterém se uzel nachází. Funkce je definována takto:

```
GetNodes (auth, node_filter, return_fields)
```

Vstupními paramtery jsou *auth* – představuje autentizační strukturu, obsahující informace o uživatelském účtu, *node_filter* – umožňuje filtrovat uzly podle zadaných kritérií, *return_fields* – seznam všech informací, které chceme získat.

### 5.1.2 SliceNodesList

Podobně jako funkce GetNodes – vrátí seznam uzlů, avšak jen ty, na kterých je instanciovaný zadaný *slice*. Využitím obou funkcí získáme seznam všech uzlů, ve kterém však můžeme odlišit ty, které jsou již instanciované daným *slicem*. U ostatních uzlů *slice* instanciovaný není – u těch avšak instanci vytvořit můžeme dodatečně. Je definována takto:

```
SliceNodesList (auth, slice_name)
```

Funkce bere jako první vstupní parametr autentizační strukturu a jako druhý parametr název *slice*.

### 5.1.3 AddSliceToNodes

S pomocí této funkce můžeme instanciovat *slice* na uzlu, na kterém ještě instance není. Změna se v centrální databázi provede okamžitě, avšak chvíli trvá, než se instance na uzlu fyzicky vytvoří.

```
AddSliceToNodes (auth, slice_id_or_name, node_id_or_hostname_list)
```

Prvním parametrem je autentizační struktura, druhým název nebo ID *slice* a třetím pole s ID nebo názvy uzlů, na kterých chceme vytvořit instanci.

### 5.1.4 UpdateSlice

S pomocí této funkce lze upravovat nastavení týkající se konkrétního *slice*. V našem případě je využita k prodloužení doby jeho platnosti. Standardně lze platnost prodloužit pouze o 2 měsíce, stačí tedy tuto funkci volat třaba jen 1x za měsíc a vždy prodloužit na maximální povolenou hodnotu. Funkce je definována následovně:

```
UpdateSlice (auth, slice_id_or_name, slice_fields)
```

Prvním parametrem je autentizační struktura, druhým název nebo ID *slice* a třetím struktura obsahující měněné parametry. Za účelem změny platnosti se nastaví ve struktuře *slice_fields* položka *expires* na nové datum platnosti.

## 5.2 Instalace SSH serveru

Pro vzdálené přihlašování je potřeba nainstalovat na každý uzel SSH server, protože není na uzlech implicitně nainstalován – přihlašování běžných uživatelů probíhá totiž přes manažera uzlu (viz 2.5.3). Po vlastní instalaci je třeba jej správně nakonfigurovat a nakonec spustit. Od té doby je jen třeba po určitých časových intervalech testovat, zdali v pořádku běží.

### 5.2.1 Instalace

Před samotnou instalací je třeba zvolit instalační program, který instalaci na uzlu provede.

Možností, jakým způsobem instalovat nové programy na uzly, je několik. Je možné využít instalační utilitu YUM nebo RPM. Oba tyto programy jsou dostupné nativně na každém uzlu PlanetLabu.

Alternativou k těmto dvěma programům může být nějaký jiný instalátor – např. již dříve zmiňovaný *Stork* v části 2.8.3. Před použitím by ovšem musel být na dané uzly sám nainstalován. Tuto možnost proto pro naše účely uvažovat nebudeme.

K výhodám, které přináší YUM oproti RPM, patří zejména to, že se jedná o automatizovaný systém, který dokáže sám vyhledat, stáhnout a nainstalovat definované balíčky, včetně balíčků závislých. Naproti tomu však instalační proces trvá déle a také spotřebovává celkem velké množství systémových prostředků.

Výhody použití systému RPM spočívají zejména v rychlosti instalace a nízkým nárokům na zdroje. Avšak závislosti balíčků je potřeba vyřešit předem, stejně tak je třeba zajistit samotné balíčky.

Implementovaný systém k instalaci SSH serveru využívá právě program RPM. Instalační skript nejprve balíček sám stáhne na uzel a poté teprv volá RPM pro jeho nainstalování. Úspěšnost instalace lze pak programově jednoduše zjistit.

### 5.2.2 Konfigurace

Před spuštěním nainstalovaného SSH serveru je potřeba jej správně nakonfigurovat.

Konfigurační soubor k využívanému SSH serveru se nachází v `/etc/ssh/sshd_config`. Nejdůležitějšími parametry jsou zejména:

- `Port: 22122 # číslo portu, na kterém server naslouchá`

  Jelikož standardní port pro SSH (22) je již obsazen SSH v manažeru uzlů (všechny porty jsou sdíleny mezi všemi *slici* na daném uzlu), je třeba tento nový SSH server spustit na jiném volném portu. Požadovaný port je možné nastavit v konfiguračním souboru řídicího systému. Systém pak tento port otestuje pomocí pokusu o otevření TCP spojení a pokud je opravdu volný, pak se použije tento. Pokud je již obsazený jiným procesem (třeba v jiném *slici* na stejném uzlu), pak se použije první další volný port. Číslo použitého portu se pak uloží i do databáze.

- `Protocol: 2 # verze protokolu SSH`

  Na výběr jsou možnosti 1 a 2. Pokud by se specifikovaly obě verze (tedy 1, 2), pak závisí na klientovi, kterou verzi použije. Ovšem bezpečnější varianta je verze 2, proto se ponechá pouze tato.

- `RSAAuthentication yes # povolit autentizaci pomocí RSA klíčů`

  Je povolena autentizace pomocí veřejného a soukromého klíče. Tato metoda je doporučena, protože je bezpečnější než autentizace heslem. Každý uživatel může do systému nahrát vlastní veřejný klíč a za pomocí soukromého klíče se pak přihlašovat.

- `PasswordAuthentication no # zakázat autentizaci uživatelským heslem`

  Autentizace přes uživatelská hesla není povolena stejně jako není povolena ani pro běžné uživatele PlanetLabu. V případě, že by měl uživatel nastavené slabé heslo nebo by jeho heslo zjistila neoprávněná osoba, byla by narušena bezpečnost uzlu.

Konfigurační soubor obsahuje ještě velké množství dalších detailních nastavení – ta se ponechají ve výchozím nastavení, které vyhovuje danému účelu.

### 5.2.3 Testování funkčnosti

Pokud je SSH server na uzlu spuštěn a je požadováno, aby byl spuštěn, je třeba po určitých časových intervalech testovat, jestli tomu tak opravdu je. Testovací skript v první fázi testuje, zdali port, na kterém má být SSH server spuštěn, je otevřený. Pokud ano, dále se netestuje, protože SSH server na daném portu naslouchá. Zrychluje to tak testování, protože se nemusí navazovat SSH spojení a spouštět příkazy na uzlu. Pokud je port však uzavřený, nebo se testování portu nezdařilo, skript se připojí přes SSH a otestuje stav SSH serveru dotazem na stav. Pokud server z nějakého důvodu neběžel, pokusí se ho spustit a situaci zaloguje.

## 5.3 Správa uživatelských účtů

Před samotným vytvářením uživatelských účtů je třeba na daném uzlu provést několik akcí, které jej potřebně nastaví:

- zavést používání stínovaného souboru `/etc/shadow` s informacemi o uživatelských účtech pro zvýšení bezpečnosti, poněvadž ten narozdíl od defaultního `/etc/passwd` je čitelný pouze uživatelem *root*. Toho lze dosáhnout použitím příkazu `pwconv` (viz 5.5.1).
- uzamknout uživatelský účet *root*. Ve výchozím nastavení je totiž bez hesla, a tudíž by každý uživatel mohl získat jeho práva použitím příkazu `su`. Tím pádem by získal administrátorský přístup do systému a mohl by mimo jiné zasahovat do nastavení jiných uživatelů, do jejich domovských adresářů a vůbec do celého nastavení systému. Avšak uzamčení tohoto účtu nemá vliv na administrátorská práva účtu pro klasické přihlašování na uzly (účet s názvem využitého *slice*), který využívá implementovaný řídicí systém, protože tento účet má práva na provádění příkazu `sudo`.
- vytvořit novou uživatelskou skupinu, která bude primární pro všechny uživatelské účty, které bude systém zavádět. Pomocí ní je pak možné všechny spravované uživatelské účty identifikovat.

Při vytváření nového účtu se každému uživateli vytvoří domovský adresář, do kterého nemá přístup žádný jiný uživatel. Po přihlášení je uživatel do tohoto adresáře automaticky přesměrován. Pokud má uživatel v době vytváření účtu v systému registrovaný RSA veřejný klíč, pak se na uzel do souboru s autorizovanými klíči zavede také, čímž je mu umožněn přístup na uzel přes jeho soukromý klíč.

Vytváření a rušení různých účtů na jednom uzlu probíhá současně při jedné příležitosti – tento proces je nazván „synchronizace“ uživatelů. Z databáze se nejprve zjistí, které účty mají na uzlu existovat a poté se zjistí, které účty na uzlu již existují. Všechy účty, které dle databáze mají na uzlu existovat, se vytvoří, pokud ještě neexistují, a všechny ostatní účty se na uzlu zruší.

Tento postup je využíván i pro kontrolu, zdali na uzlu existují přesně jen ty účty, které mají.

## 5.4 Technologie

Řídicí část celého systému tvoří skripty spouštěné v prostředí Linuxu. Tyto skripty jsou napsány v jazyce PHP, které je spouštěno v režimu CLI (Command Line Interface) – tedy v režimu příkazové řádky, určenému právě pro skriptování. Tato technologie byla zvolena především kvůli využití několika existujících komponent, které současně využívá PHP ve správní části řídicího systému. Těmi jsou:

- knihovna pro přístup k MySQL databázi
- knihovna implementující protokol XML-RPC
- PHP modul implementující protokol SSH

### 5.4.1 XML-RPC

XML-RPC [13] je protokol, umožňující volat vzdálené procedury přes Internet. Využívá pro přenos dat protokol HTTP a pro kódování dat XML. XML-RPC zpráva je HTTP-POST požadavek na vykonání procedury na serveru. Ten proceduru vykoná a zašle zpět HTTP zprávu obsahující odpověď s daty kódovanými opět v XML. Výhoda tohoto protokolu spočívá v nezávislosti na architektuře a operačním systému.

Ve skriptech, využívajících PlanetLab API, je použita volně dostupná knihovna implementující XML-RPC protokol pro PHP [12].

### 5.4.2 SSH

Pro vzdálený přístup k uzlům PlanetLabu se využívá protokol SSH. Aby se mohly na vzdálené servery připojovat řídicí skripty, bylo zapotřebí nainstalovat do PHP modul s názvem SSH2, který pak poskytuje široké spektrum užitečných funkcí. Mezi ty základní patří:

- `ssh2_connect` – připojení se na server
- `ssh2_auth_pubkey_file` – autentizace užitím veřejného a soukromého klíče
- `ssh2_exec` – vykoná příkaz na vzdáleném serveru
- `ssh2_scp_send` – odešle soubor přes protokol SCP

Pro práci s těmito funkcemi byla navržena třída `ssh_pubkey_connection`, která zpřehledňuje a zjednodušuje práci s těmito funkcemi, a poskytuje tak efektivní způsob pro správu vzdálených serverů.

## 5.5 Využití příkazů shellu Linuxu při řízení uzlů

PHP skripty využívají k dosahování svých cílů sadu funkcí, které jsou rozděleny do několika skupin, podle svého účelu takto:

- vstupně-výstupní funkce – funkce pro čtení a zápis dat z/do souborů
- databázové funkce – funkce pro manipulaci s daty uloženými v databázi

- funkce pro manipulaci s uživatelskými účty
- funkce pro instalaci programů a jejich manipulaci – využity pro instalaci SSH serverů

Poslední dvě jmenované využívají SSH spojení a na připojeném uzlu vykonávají příkazy linuxového shellu. Právě tyto příkazy fyzicky manipulují se vzdáleným uzlem.

Většina těchto příkazů vyžaduje práva *root* pro jejich spuštění. Toho se dosahuje tím, že jsou volány pomocí příkazu `sudo`. Některé příkazy také vyžadují zadat absolutní cestu k nim, poněvadž implicitně nemusí být nastavena v proměnné prostředí PATH. Většina z nich se nachází v adresáři `/usr/sbin/`.

### 5.5.1 Manipulace s uživatelskými účty

V této části jsou popsány a vysvětleny příkazy shellu, které řídicí systém využívá ve svých skriptech pro vzdálenou správu uživatelských účtů.

- **groupadd** – vytvoření nové uživatelské skupiny

  ```
  groupadd student
  ```

  – vytvoří uživatelskou skupinu *student*.

  Systém tento příkaz používá v případě, že na daném uzlu ještě neexistují žádní uživatelé a tato skupina neexistuje. Tato skupina je pak primární pro všechny uživatele vytvářené systémem a podle ní jsou pak identifikováni.

- **useradd** – vytvoření nového uživatele

  ```
  useradd xnovak01 -g student -p passwordhash
  ```

  – vytvoří uživatele *xnovak01*, jehož primární uživatelskou skupinou je skupina *student* a jehož heslo je uloženo formou haše MD5 *passwordhash*.

  Takto je vytvářen každý nový uživatelský účet na uzlu. Systém ho přiřadí do definované uživatelské skupiny a nastaví mu heslo z databáze, které před použitím tohoto příkazu zahašuje algoritmem MD5. Heslo se hašuje spolu s tzv. solí, což je náhodný textový řetězec, díky němuž pak výsledný haš pro stejné heslo s různou solí vypadá jinak. Tato sůl je pak uložena před samotným hašem a je znovu použita, pokud se ověřuje heslo. Princip „solení“ hašů je bezpečnostní mechanismus, který ztěžuje slovníkové útoky na haše.

- **userdel** – smazání uživatelského účtu

  ```
  userdel -r xnovak01
  ```

  – smaže uživate *xnovak01*. Parametr `-r` způsobí, že bude smazán i jeho domovský adresář včetně všech souborů uvnitř.

  Systém tento příkaz využívá, pokud na uzlu existuje nějaký uživatelský účet, který není registrovaný v databázi k tomuto uzlu.

- **pwconv** – vytvoření stínovaného souboru `/etc/shadow` ze souboru `/etc/passwd`

  Použije se před vlastním vytvářením uživatelů na uzlech. Zajistí tak, že soubor s hesly není čitelný pro běžné uživatele, což zvyšuje bezpečnost systému.

- **passwd** – změna uživatelského hesla

  ```
  passwd -l root
  ```

  – uzamče účet *root* tím, že nastaví heslo na hodnotu, která se nemůže shodovat s žádnou možnou zašifrovanou hodnotou.

  Toto opatření se provádí proto, aby se uživatelům odepřela superuživatelská práva, díky nimž by mohli zasahovat do nastavení systému a do účtů jiných uživatelů.

  ```
  passwd -u -f root
  ```

  – odemče účet *root*. Parametr `-f` umožní, aby se účet odemčel i přesto, že měl před uzamčením nastavené prázdné heslo.

  Tento příkaz se použije ve chvíli, kdy systém přestane využívat daný uzel a vyřazuje ho ze skupiny spravovaných uzlů.

- **cat** – konkatenace souborů a tisk na standardní výstup

  ```
  cat /etc/group | grep student
  ```

  – vypíše název uživatelské skupiny *student*, pokud v systému existuje.

  Takto systém testuje, jestli je daná uživatelská skupina v systému vytvořena. Testování se provádí vždy, když jsou vytvářeny nové uživatelské účty. Pokud tato skupina neexistuje, tak je následně vytvořena, pokud existuje, tak se nevytváří.

- **awk** – jazyk pro zpracování textových dat

  Systém využívá příkazy AWK pro zjištění existujících uživatelských účtů na uzlu. Účty jsou identifikovány na základě jejich primární uživatelské skupiny. Pokud je touto skupinou skupina *student*, pak se příkazem

  ```
  GID=‘awk -F: ’$1 == "student"  {print $3}’ /etc/group‘
  ```

  uloží do proměnné GID číslo této skupiny ze souboru `/etc/group` a následně se příkazem

  ```
  awk -v gid=$GID -F: ’$4 == gid {print $1}’ /etc/passwd
  ```

  vypíšou ze souboru `/etc/passwd` všechny uživatelské účty, které do této skupiny primárně patří.

### 5.5.2 Instalace programů a jejich manipulace

V souvislosti s instalaci programů na uzly, především za účelem instalace SSH serveru, využívá řídicí systém tyto příkazy:

- **wget** – program pro stahování souborů ze sítě

  ```
  wget http://path-to-openssh-server.rpm
  ```

  – stáhne RPM balíček openssh-server ze zadané url adresy

  Takto systém stahuje balíček např. s SSH serverem nebo balíček `tcp_wrappers`, který je nezbytný pro instalaci SSH serveru.

- **rpm** – správce balíčků

  `rpm -q openssh-server`

  – zjistí, zdali je v systému nainstalován openssh-server

  Před instalací SSH serveru se takto zjišťuje, jestli už není SSH server na uzlu nainstalován.

  `rpm -i openssh-server-4.2p1-fc4.10.i386.rpm`

  – nainstaluje zadaný balíček

  Na každý nový uzel, který je přidán do systému, se po stažení automaticky nainstaluje balíček obsahující SSH server, který je následně nakonfigurován a spuštěn.

  `rpm -e openssh-server`

  – odinstaluje zadaný balíček

  V případě, že je ze systému vyřazován některý uzel, tak se takto odinstaluje SSH server.

- **/etc/init.d/sshd** – démon nainstalovaného SSH serveru

  Parametry, kterými manipulujeme s SSH serverem jsou :

  `sshd status`

  – zjistí stav SSH serveru

  Dotazem na stav se zjišťuje, zdali SSH server v pořádku běží nebo jestli je zastaven.

  `sshd start`

  – spustí SSH server

  Po úspěšné instalaci se tímto příkazem spouští SSH server nebo v případě, že byl zastaven a je požadavak na jeho spuštění.

  `sshd stop`

  – pozastaví SSH server

  Využívá se při požadavku na vypnutí SSH serveru.

- **ssh-keygen** – program na generování autentizačních klíčů a jejich správu

  `ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub`

  – vygeneruje otisk (fingerprint) RSA klíče

  Systém si ukládá otisky veřejných klíčů ze všech SSH serverů, aby si přihašující se uživatelé mohli ověřit identitu serveru, ke kterému se připojují. Zabrání se tak možnému podvržení identity serveru.

## 5.6 Bezpečnost

Na bezpečnost je brán velký zřetel v celém PlanetLabu. Je navržen tak, aby zajistil maximální bezpečnost, ale současně aby nebylo složité k němu přistupovat pro oprávněné uživatele.

Snaha o maximální bezpečnost, a přitom také o rozumnou míru použitelnosti, je uplatněna také v implementovaném systému. Ten se inspiruje právě v samotném PlanetLabu. Bezpečnost se týká všech součástí systému. Jakmile by jedna z nich selhala v bezpečnosti, nastalo by ohrožení kompromitace celého systému. Proto je třeba zajistit, aby všechny součásti systému byly bezpečné samy o sobě.

Jedná se tedy o bezpečnost řídicího systému – webserveru, s kterým komunikují uživatelé a administrátor, samotné databáze, jejíž kompromitace by byla fatální pro celý systém, poněvadž je to úložiště všech důvěrných dat, a SSH serverů na uzlech, jejichž špatná konfigurace by mohla vést k neoprávněnému přístupu na uzly.

Avšak pro zajištění bezpečnosti nestačí pouze správné nastavení systému, je také potřeba, aby se jeho uživatelé chovali odpovědně a nedopustili tak svým jednáním vyzrazení osobních přístupových údajů nebo aby nějakým způsobem nenarušili bezpečný chod systému.

### 5.6.1 Web

Na webový server je umožněn přístup pouze přes protokol HTTPS – tedy protokol zabezpečený pomocí bezpečnostního protokolu SSL. To znamená, že veškerá komuninakce uživatele s webserverem je šifrována, a tedy při autentizaci uživatele k řídicímu systému není možné odchytit zadávané přístupové heslo.

Všechny webové stránky (kromě přihlašovací stránky a stránky pro registraci uživatelů) jsou přístupné pouze autentizovaným uživatelům a nehrozí tedy, že by se k údajům a nastavení systému dostala neautorizovaná osoba – bez znalosti hesla. Tento princip je řešen využitím *PHP sessions*, kdy se po přihlášení uživatele nastaví proměnná session s loginem uživatele, a ta je pak v záhlaví každé stránky ověřována.

### 5.6.2 Databáze

Aby byla databáze maximálně bezpečná, bylo by třeba provést velké množství různých bezpečnostních opatření, poněvadž existuje mnoho různých útoků na databázové servery, které využívají nejrůznějších slabin v jejich nastavení, zejména v tom výchozím.

Použitá databáze (MySQL) je zajištěna zejména proti těm zásadnějším bezpečnostním rizikům.

- Databázový server je dostupný pouze z *localhostu*, takže připojovat se na něj nelze vzdáleně
- V databázi existují pouze nezbytné uživatelské účty, které mají nastavená silná hesla
- Oprávnění účtu, který pro přístup využívá PHP, má omezená práva – pouze na využívanou databázi „planetlab“ s oprávněním vykonávat nad ní pouze operace SELECT, INSERT, UPDATE a DELETE

Další možná opatření, která by mohla být nastavena pro zvýšení bezpečnosti jsou:

- spouštět databázový server v *chrootovaném* prostředí – to by zajistilo, že by měl server fyzicky přístup pouze do vlastní souborové struktury

- přejmenovat administrátorský účet *root* – útočník by nový název neznal a neúspěšně by se snažil přihlašovat jako *root*
- mazat soubory s historií SQL dotazů

### 5.6.3 SSH server

Zabezpečení přístupu na uzel je dáno konfigurací SSH serveru, který na něm běží. Nainstalovaný SSH server je totiž jedinným možným bodem vzdáleného přístupu na uzel, pokud neuvažujeme přístup běžných uživatelů, kde bezpečnost přístupu má na starosti PlanetLab.

Nainstalovaný SSH server povoluje autentizaci pomocí RSA klíčů. To znamená, že přistupující uživatel potřebuje mít svůj soukromý a veřejný klíč. Tento pár lze vygenerovat pomocí linuxového příkazu `ssh-keygen`. V SSH verzi 2, která je vyžadována pro přístup, se soubor se soukromým klíčem jmenuje `id_rsa` a soubor s veřejným klíčem `id_rsa.pub`. Jedná se o textové soubory, v nichž je binární klíč reprezentován textově pomocí kódování *Base64*. Tyto soubory se běžně ukládají v domovském adresáři uživatele do složky `.ssh`.

K soukromému klíči nesmí mít přístup nikdo jiný než jeho vlastník. Kdyby se k němu dostala jiná osoba, byla by narušena bezpečnost a uživatel by si měl vytvořit nový klíčový pár. Naproti tomu veřejný klíč může být vystaven pro ostatní. Bezpečnost soukromého klíče lze ještě zvýšit tím, že jej necháme zašifrovat. Již při vytváření klíčů je možné v programu `ssh-keygen` zvolit možnost zašifrovat soukromý klíč. Při této volbě je nutné zvolit si heslo, pomocí kterého se bude šifrovat. Tato volba je doporučena, protože klíč je pak chráněn silnou symetrickou šifrou, ke které zná heslo pouze jeho vlastník. Pokud šifrování není zvoleno při vytváření klíče, je možné zašifrovat klíč kdykoliv později příkazem `ssh-keygen -p`.

Svůj veřejný klíč uživatel nahraje přes správní systém do databáze, odkud je následně z výkonné části zkopírován na všechny uzly, na kterých má uživatel svůj účet. Na uzlech je klíč nahrán v domovském adresáři uživatele do souboru `.ssh/authorized_keys`, což znamená pro SSH server, že daný uživatel se může SSH serveru autentizovat pomocí soukromého klíče, který je párový k uloženému veřejnému klíči.

Jestliže si uživatel vytvoří nový klíčový pár, může nový veřejný klíč nahrát do databáze a výkonná část jím nahradí původní klíč na uzlech. Uživatel pak bude k uzlům moci přistupovat pomocí nových klíčů.

Aby si uživatel při prvním přihlášení na uzel mohl ověřit, že se přihlašuje opravdu na ten správný (a že tedy někdo nepodvrhl identitu tohoto uzlu), má k dispozici ve správní části systému informace o otiscích (fingerprints) RSA veřejných klíčů ke všem přiřazeným uzlům. Když se pak na daný uzel připojuje, může si jeho identitu zkontrolovat porovnáním otisku serveru, na který se přihlašuje, a otisku uloženého v systému. Pokud jsou otisky stejné, má uživatel jistotu, že se připojuje na správný uzel. Jeho veřejný klíč se mu automaticky uloží do souboru `known_hosts` se známými klíči a při příštím přihlášení je pak ověřování klíče prováděno automaticky právě kontrolou tohoto souboru.

Vzhledem k zavedeným bezpečnostním mechanizmům vzdáleného připojování by bezpečnost uzlů měla být dosti vysoká a při dodržení bezpečnostních postupů uživateli by měly být uzly dobře chráněny před neautorizovaným přístupem.

## 5.7 Výsledek implementace

Cílem implementace bylo realizovat navržený systém, který by umožňoval přístup na uzly PlanetLabu skupině uživatelů, kteří sami nejsou registrováni v organizaci PlanetLab, s

využitím jednoho registrovaného uživatelského účtu.

Tento systém byl realizován a nasazen v prostředí operačního systému Linux (Ubuntu Server Edition 7.10) ve formě obrazu (image) virtuálního počítače pro virtualizační software VMware Server. Hlavní důvody pro využití virtualizačních nástrojů byly:

1. možnost konfigurace celého operačního systému na míru danému účelu
2. snadná přenositelnost systému mezi různými fyzickými počítači
3. oddělení implementovaného systému od jiných běžících systémů a tím nenarušení vzájemné bezpečnosti
4. neplýtvání hardwarem – za pomoci virtualizace je možné mít spuštěných více virtuálních strojů na jednom fyzickém

Implementovaný systém využívá tyto nainstalované aplikace pro své různé části:

- PHP5 s modulem SSH2 – správní i výkonná část
- webový server Apache2 – správní část
- databázový server MySQL – perzistentní část
- plánovač Cron – výkonná část

Plánovač úloh Cron je využit k pravidelnému spouštění řídicích skriptů systému. Má naplánováno spouštět všechny řídicí skripty v přesně stanovených časových intervalech. Tyto skripty se pak snaží stav uzlů synchronizovat s požadovanými stavy uloženými v databázi. Změny tak nejsou prováděny ihned po nastavení administrátorem, ale až ve chvili, kdy je spuštěn plánovaný skript.

Cron a jeho naplánované úlohy se nastavují editací konfiguračního souboru pomocí příkazu `crontab -e`. Výpis tohoto konfiguračního souboru lze provést příkazem `crontab -l`. Definované časové intervaly spouštěných skriptů se tak dají případně změnit (častější nebo méně časté spouštění), pokud by nevyhovovalo současné nastavení.

# Kapitola 6

# Závěr

Cílem této diplomové práce bylo navrhnout a následně realizovat systém, který by umožňoval vytvářet a spravovat virtuální uživatelské účty na vybraných uzlech PlanetLabu, tj. účty, které všechny využívají jednoho společného registrovaného účtu, který máme k dispozici.

Ze dvou zajímavých řešení jsem zvolil to, které se zdálo pro daný účel použitelnější a u kterého jsem byl více přesvědčen, že je možné úspěšně je realizovat.

Výsledným produktem je obraz (image) operačního systému ve virtuálním počítači pro virtualizační software VMware Server s nainstalovaným a nakonfigurovaným implementovaným systémem, takže je jednoduše nasaditelný a je možné téměř hned po spuštění obrazu začít systém provozovat.

Systém v současné době zajišťuje jednak správu uživatelů na uzlech a jednak mechanismus, přes který se na ně mohou připojovat. Tento mechanismus je realizován instalací SSH serverů. Jako případné další rozšíření by bylo možné vytvořit mechanismus, který by umožňoval instalovat a spravovat obecně jakýkoli software pro uživatele na uzlech. Pro tento účel by bylo možné využít již implementovaného programového mechanismu připojování se na uzly a některé vytvořené funkce pro instalaci softwarových balíčků. Druhou variantou by bylo využít již existujících programů, které správu instalovaného softwaru provádějí, a nějak je propojit s tímto systémem, aby se správa uživatelů a softwaru sjednotila do jednoho systému.

Výsledný systém byl od počátku navrhován pro potřeby FIT VUT v Brně, které má k PlanetLabu přístup pouze jako uživatel, nikoliv správce (PI). Systém tak vytváří roli správce virtuálních uživatelů, jež je určena vyučujícímu na fakultě, a roli samotných virtuálních uživatelů, která patří studentům.

Přínos projektu spočívá tedy zejména v samotném implementovaném systému, který umožní studentům přístup na uzly PlanetLabu a vyučujícím poskytne možnost, jak tento přístup řídit.

Poněvadž systém nebyl nasazen do ostrého provozu, nemohl být potřebně dlouho a při reálném využití otestován. Je proto možné, že obsahuje i nějaké nedostatky, na které by se přišlo až při opravdovém nasazení, stejně jako konfigurace řídicí části (zejména frekvence spouštění skriptů) by se měla dodatečně doladit s ohledem na konkrétní nasazení.

# Literatura


[1] *CoDNS project*. [online], [cit. 2007-12-16].
URL <http://codeen.cs.princeton.edu/codns/>

[2] *CORAL: The Coral Content Distribution Network*. [online], [cit. 2007-12-16].
URL <http://www.coralcdn.org/>

[3] *PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services*. [online], [cit. 2007-12-15].
URL <http://www.planet-lab.org>

[4] *Plkmod: SILK in PlanetLab*. [online], [cit. 2007-12-16].
URL <http://www.cs.princeton.edu/~acb/plkmod/>

[5] *Linux VServers Project*. [online], [cit. 2007-12-16].
URL <http://linux-vserver.org>

[6] *PlanetLab Application Manager*. [online], [cit. 2008-05-02].
URL <http://appmanager.berkeley.intel-research.net/>

[7] *PlanetLab Central API Documentation*. [online], [cit. 2007-12-17].
URL <http://www.planet-lab.org/doc/plc_api>

[8] *pShell: An Interactive Shell for Managing Planetlab Slices*. [online], [cit. 2008-05-02].
URL <http://www.cs.mcgill.ca/~anrl/projects/pShell/>

[9] *pssh*. [online], [cit. 2008-05-02].
URL <http://www.theether.org/pssh/>

[10] *Stork – software installation utility*. [online], [cit. 2008-05-02].
URL <http://www.cs.arizona.edu/stork/>

[11] *VNET: PlanetLab Virtualized Network Access*. [online], [cit. 2007-12-16].
URL <http://www.planet-lab.org/doc/vnet>

[12] *XML-RPC for PHP*. [online], [cit. 2008-05-02].
URL <http://phpxmlrpc.sourceforge.net/>

[13] *XML-RPC Specification*. [online], [cit. 2008-05-02].
URL <http://www.xmlrpc.com/spec>

[14] BAVIER, A.; BOWMAN, M.; CHUN, B.; aj.: Operating System Support for Planetary-Scale Network Services. 2004.

[15] BAVIER, A.; VOIGT, T.; WAWRZONIAK, M.; aj.: SILK: Scout Paths in the Linux Kernel. 2002.

[16] KUBIATOWICZ, J.; BINDEL, D.; CHEN, Y.; aj.: OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*, ACM, November 2000.

[17] PETERSON, L.; ANDERSON, T.; CULLER, D.; aj.: A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of the 1st ACM Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.

[18] PETERSON, L.; ROSCOE, T.: The Design Principles of PlanetLab. *Operating Systems Review*, ročník 40, č. 1, January 2006: s. 11–16.

# Dodatek A

# Obsah přiloženého DVD

Součástí této práce je DVD nosič, který obsahuje elektronickou verzi této technické zprávy, zdrojové kódy implementovaného systému, návod, jak systém použít a obraz operačního systému virtuálního počítače pro virtualizační software VMware Server, který obsahuje nasazený systém, který je možné po spuštění obrazu ihned využívat.