VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

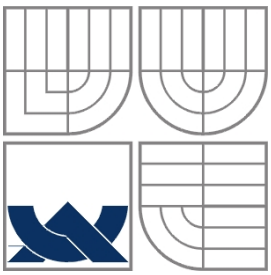
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE KLIENT/SERVER V GLOBÁLNÍ POČÍTAČOVOU SÍŤ PLANETLAB

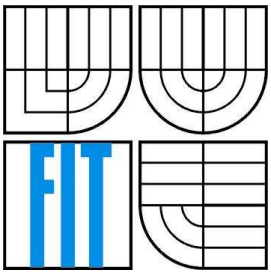
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE MILAN SLADKÝ
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

APLIKACE KLIENT/SERVER V GLOBÁLNÍ POČÍTAČOVOU SÍŤ PLANETLAB

CLIENT/SERVER APPLICATION IN GLOBAL COMPUTER NETWORK PLANETLAB

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

MILAN SLADKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MATOUŠEK, Ph.D.

BRNO 2008

Abstrakt

Předmětem této bakalářské práce je seznámit se s experimentální sítí PlanetLab, její historií, vývojem, principem fungování a využitím této sítě v praxi, například pro vývoj nových síťových architektur. Programová část bakalářské práce zahrnuje návrh a implementaci vhodné aplikace typu klient/server, která pracuje v prostředí PlanetLab.

Klíčová slova

PlanetLab, slice, uzel, Internet, síť, testbed, overlay, multicast, distribuovaná virtualizace

Abstract

Objective of this bachelor's thesis is to get acquaint with experimental network PlanetLab, its history, development, principle of its behaviour and usage of this network, for example for development of new network architectures.
Program part of this bachelor's thesis consists of concept and implementation of suitable client/sever type application, which runs on PlanetLab network.

Keywords

PlanetLab, slice, node, Internet, network, testbed, overlay, multicast, distributed virtualisation

Citace

Sladký Milan: Aplikace klient/server v globální počítačovou síť PlanetLab. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Aplikace klient/server v globální počítačovou síť PlanetLab

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petra Matouška, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Milan Sladký
Datum

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Ing. Petru Matouškovi, Ph.D. za čas, ochotu a rady při konzultacích. Dále bych chtěl poděkovat mým blízkým za pomoc a podporu.

Obsah

Úvod

Internet je pro většinu lidí pohybujících se v oblasti informačních technologií věc naprosto samozřejmá. Poslední dobou se však stále více hovoří o problémech, které se objevují v souvislosti s rozšiřováním a růstem Internetu v celém světě. Myšlenka a princip fungování Internetu je starý desítky let a to v dnešní době přináší mnohá úskalí, která bude dříve či později potřeba vyřešit. Mezi ty nejpalčivější problémy patří omezený a již nedostačující adresový prostor, nespolehlivost a nedostatečnost přenosových cest, útoky na servery a další neméně vážné problémy. Vzhledem k již zmiňovanému stáří celé koncepce (mnohé služby a protokoly jsou staré desítky let) fungování Internetu není možné tuto koncepci náhle změnit, ale pouze postupně modifikovat.

Právě tyto problémy a otázky daly vzniknout organizaci PlanetLab. Organizaci, která má za svůj cíl změnu Internetu. Na následujících stranách této technické zprávy se pokusím vysvětlit, co je vlastně organizace PlanetLab, v jakých oblastech se snaží působit, jaké možnosti nabízí a jaké aplikace je možné testovat a vyvíjet za pomoci této organizace a její sítě.

V první kapitole se snažím popsat problémy dnešního Internetu a dostupné prostředky a návrhy, kterými je zatím možné tyto problémy řešit. Jedním z možných řešení těchto problémů je právě návrh, který přináší PlanetLab. Jednotlivé koncepce a návrhy jsou zde popsány a zhodnoceny.

Následující kapitola popisuje samotný PlanetLab. Je zde popsán princip fungování koncepce PlanetLabu. Dále je zde popsána základní terminologie PlanetLabu, podmínky členství a možnosti užívání.

Ve třetí kapitole je popsána technická specifikace PlanetLabu. Jsou zde zmíněny všechny důležité komponenty, ze kterých se PlanetLab po technické stránce skládá. Jednotlivé komponenty jsou popsány včetně výhod a omezení, které přináší.

Čtvrtá kapitola popisuje možnost vytváření aplikací a práci s PlanetLabem z pohledu koncového uživatele. Pro demonstraci práce s PlanetLabem je určena jedna z příloh, která obsahuje jednoduchého průvodce vytvořením a spuštěním aplikace.

Následující kapitola popisuje případové studie užití PlanetLabu a možnost realizace těchto studií.

Poslední kapitola popisuje návrh a implementaci aplikace, která běží v prostředí PlanetLabu. Jedná se o návrh a implementaci multicastu.

V závěru hodnotím práci a dosažené výsledky a zmiňuji možnosti, ke kterým by se dal PlanetLab dále využívat.

1 Problémy dnešního a budoucího Internetu

Jak již bylo v úvodu řečeno, Internet se jako takový potýká s řadou problémů a je potřeba začít tyto problémy rychle řešit, protože jinak Internetu hrozí nevyhnutelný kolaps. Je velmi malá naděje, že dojde k zásadním změnám v architektuře a principu fungování celé sítě. Navíc je tu další velký problém, že ne všichni poskytovatelé Internetu by přistoupili k realizaci těchto změn.

Neschopnost adaptovat se novým požadavkům dala vzniknout řadě ad-hoc řešení, která dále narušují architektu Internetu a celou situaci dále zhoršují. Tato řešení jsou navržena s cílem splnit konkrétní požadavek, který samotná architektura Internetu neumožňuje. Jsou to krátkodobá a většinou jednoúčelová řešení, ale celkově poškozují dlouhodobou flexibilitu, spolehlivost a možnosti správy celého Internetu.

Přes všechna tato úskalí vývoj a výzkum pokračuje, avšak samotný vývoj Internetu je mnohem více ovlivňován komerčními zájmy než vědeckým výzkumem. Nová síťová architektura by měla být orientována na služby, které může poskytnout. Internet by měl být definován službami, nikoli možnostmi samotného přenosu dat. Ukázalo se však, že experimentování s novými architekturami a principy je poněkud více problematické, než by se na první pohled mohlo zdát. Testování těchto nových principů, architektur, případně konkrétních aplikací může probíhat za pomoci simulace, emulace, testbeds (platforma, na které je možné vyvíjet nástroje a produkty, které mohou být nasazeny a mohou komunikovat v reálném čase) a overlays (v případě sítí jde přesněji o overlay síť; overlay síť je vybudována na jiné síti, která může být overlay síť vůči jiné síti). Jednoduchým, velmi obecným, ale zato velmi názorným příkladem overlay sítě je Internet, který funguje po běžných telefonních linkách na vytáčeném spojení.

1.1 Testbeds a overlays

Pokud budeme chtít nasadit nebo jen uvažovat o nasazení nově navržené architektury, je nejdříve nutné tuto architekturu otestovat a zhodnotit. Simulace a emulace jsou jistě užitečné nástroje pro tyto účely, nemohou však v žádném případě nahradit experimenty s reálným síťovým provozem.

Při samotné implementaci daného návrhu je potřeba řešit nepříjemné fakty, které se běžně neřeší a opomíjejí. Mezi tyto fakty lze zařadit dědičnou síť (původní název legacy network; je to síť, která není založena na IP protokolu), velký počet různých poskytovatelů, různé anomální kolize a selhání na sítí, případně anomální chování a nároky aplikací a jistě i další problémy. Reálný síťový provoz navíc poskytuje komplexnější obraz o tom, jak se bude daná architektura chovat a zda bude mít i tolik požadovaný přínos v dané oblasti.

Pro experimenty a testování nových architektur se používají např. fyzické testbeds a overlays. Každý z těchto přístupů má ovšem svá pro a proti.

1.1.1 Fyzické testbeds

Jedná se o tradiční platformu pro experimentování a testování. Ve své podstatě jde o několik lokací s určitým výpočetním výkonem, které jsou mezi sebou propojeny pronajatými linkami. Daly by se rozdělit podle zaměření – na ty, které jsou orientované na výzkum a pak na produkčně orientované.

1.1.1.1 Produkční testbeds

Takovéto testbeds, jako je např. Internet2, podporují a vytváří skutečný síťový provoz, který vzniká skutečnými uživateli a to i ve velkém množství a na mnoha místech. Díky tomu lze zjistit cenné informace o chování dané architektury. Ale vzhledem k tomu, že se jedná o produkční a nikoli výzkumně orientovaný testbed, tak většina uživatelů nemá ani tušení, že je jejich síťový provoz používán pro experimentování a zjišťování chování dané architektury. Tito uživatelé očekávají výkon, spolehlivost a stabilitu, která nemůže být v žádném případě horší, než v normálním Internetu. Pokud by byl např. výkon horší, nemělo by právě pro tyto uživatele takovou síť, založenou na dané architektuře, smysl používat. Z tohoto důvodu musí být experimenty prováděné v produkčních testbeds velmi konzervativní.

1.1.1.2 Výzkumné testbeds

U tohoto typu testbeds, jako je např. DETER, se nevyskytuje skutečný síťový provoz od skutečných uživatelů. Místo toho bývá většinou použit uměle generovaný provoz nebo provoz od zúčastněných uživatelů, případně kombinace obojího. Díky tomu experimenty nemusí být konzervativní, ale naopak je možné testovat zcela nové návrhy. Ovšem velkou nevýhodou je již zmiňovaná absence skutečných uživatelů a skutečného síťového provozu, protože bez těchto dvou aspektů pak není možné otestovat životaschopnost v reálném provozu.

1.1.1.3 Využití fyzických testbeds

Když porovnáme všechna pro a proti každé z těchto koncepcí, tak zjistíme, že se ani jedna z těchto koncepcí svými vlastnostmi vyloženě nehodí k testování a hlavně zhodnocení celkové funkčnosti a přínosu zkoumané architektury či návrhu.

Pokud budeme navíc zvažovat fakt, že obě koncepce vyžadují dedikované linky, které nejsou levné, tak zjistíme, že provozovat takovou koncepci v opravdu velkém měřítku by bylo extrémně nákladné.

1.1.2 Overlays

I přes relativně nedávné nasazení se overlays staly rozšířeným prostředkem pro experimenty a provoz již realizovaných návrhů. Nejsou geograficky limitovány a jejich použití je dobrovolné. Navíc se použití overlays zpravidla nevyznačuje vysokými náklady, což se nedá říci o použití fyzických testbeds.

Overlays byly a jsou používané ve velkém měřítku jako omezená řešení specifických problémů dnešního Internetu. Může se jednat o lepší výkon, dostupnost služeb, distribuci obsahu nebo třeba multicast. Každý problém vyžadoval rychlé řešení a takovéto řešení bylo téměř vždy navrženo izolovaně pouze pro daný problém. Tato řešení, byť na sobě nezávislá, jsou nasazována do běžného provozu, aniž by bylo řádně odzkoušeno, jak se jednotlivá řešení budou vzájemně ovlivňovat a chovat. Místo navrhování takovýchto jednoúčelových a krátkozrakých řešení by bylo mnohem přínosnější vytvořit řešení pomocí sady overlays, která by nahradila zastaralou architekturu Internetu. Mezi typické overlay řešení patří peer-to-peer sítě, jako je např. Skype, KaZzA, WinMX a další.

Pokud budou overlays využívány hlavně prvně zmíněným způsobem, stane se z nich jakási pomocná berlička pro řešení něčeho, na co už samotná architektura Internetu nestačí. Druhý zmiňovaný způsob by mohl znamenat fundamentální změny v samotné architektuře Internetu. Overlays by měly být chápany a používány jako možnost nového přístupu a pohledu na věc, než technická vymoženost, kterou můžeme využít pro každý jednotlivý problém.

1.1.3 Virtuální testbeds

Z výše uvedeného můžeme vidět klady a zápory zmíněných řešení. Jako reakce byla navržena koncepce Virtual Testbed. Jedná se o platformu, na které je možné provádět vědecké experimenty, ale zároveň je možné v ní nasadit a provozovat běžné aplikace pro běžné uživatele. Virtual Testbeds se skládají ze dvou hlavních částí: overlay vrstva a klient - proxy mechanismus.

1.1.3.1 Overlay vrstva

Overlay vrstva poskytuje systém oddělených, ale zároveň multiplexních overlay uzlů. Díky tomu, že jsou uzly multiplexní, je možné provozovat mnoho různých experimentů na jedné a té samé infrastruktuře (tato možnost je poprvé uvedena do praxe právě v PlanetLabu).

1.1.3.2 Klient – proxy mechanismus

Tento mechanismus považuje další uzel za první následující router, bez nutností zavádět jakákoli další omezení na této experimentální architektuře. Tento mechanismus také umožňuje směrovat lokální provoz přímo nebo případně rozhodovat podle požadavků konkrétních aplikací. Tyto dvě možnosti řeší problém bariéry přístupu a také řeší omezení z hlediska architektury, se kterými se overlays musely potýkat (opět je tato možnost uvedena poprvé do praxe právě v PlanetLabu).

1.2 PlanetLab

Pokud budeme chtít přesně říci, co PlanetLab (http://www.planet-lab.org) vlastně je, tak použijeme pojem virtual overlay testbed. Z výše uvedených si PlanetLab bere to nejlepší a snaží se tyto myšlenky uvést do praxe. Cílem PlanetLabu je definovat nový standart Internetu, konkrétně se jedná o NGI (Next Generation Internet). V následujících kapitolách je popsáno a rozebráno, jakým způsobem PlanetLab přesně funguje a jakými způsoby chce dosáhnout svých dozajista vysokých sílů.

1.3 Ostatní testbeds a overlays

Do dnešních dní samozřejmě vznikla celá řada testbeds a overlays s určitými kladnými i zápornými vlastnostmi.

1.3.1 Internet2

Internet2, který zahrnuje páteř Abilene, je fyzická vysokorychlostní optická síť, která spojuje největší univerzity v USA. Jednotlivé uzly jsou uzavřené komerční routery, takže je téměř nemožné zavádět nové a neodladěné experimentální projekty do této sítě.

Kontakt a členství

Veškeré možné dostupné informace je možné nalézt na webových stránkách http://www.internet2.edu. Členem se nemůže stát fyzická osoba. Je potřeba patřit pod nějakou organizaci, která již může být členem. Členství je vždy placené, i v případě, že se jedná např. o univerzitu.

1.3.2 Emulab

Emulab je experimentální síť podporovaná University of Utah. Je možné experimentovat se sadou uzlů a konfigurovat je pro emulaci různých síťových topologií. Využití této overlay pro jiné experimenty je také možné. Jedná se o platformu, která slouží k testování a nasazení nových síťových technologií a architektur, takže většina projektů běží dlouhodobě a výsledky projektů jsou průběžně analyzovány.

Kontakt a členství

Veškeré dostupné informace lze nalézt na webových stránkách http://www.emulab.net. Členství není nijakým způsobem zpoplatněno a členem se po splnění podmínek uvedených na výše zmíněných internetových stránkách může stát v podstatě každý, kdo spadá pod nějakou organizaci (např. univerzita).

1.3.3 XBONE

XBONE je overlay síť, která podporuje IP-in-IP tunelování. Navíc obsahuje sadu nástrojů s vlastním GUI, které slouží k nastavování a monitorování jednotlivých konfigurací overlay. XBONE umožňuje vytvoření více overlays na fyzicky totožných strojích. Všechny tyto overlays jsou na sobě nezávislé, ale limitované možnostmi IP tunelování, což limituje možnost použití overlay vyšší úrovně (použité např. u peer-to-peer sítí).

Kontakt a členství

Informace o tomto projektu je možné nalézt na stránkách http://www.isi.edu/xbone/, který provozuje University of Southern California's Information Sciences Institute. Podle neaktuálnosti webu by se dalo říci, že projekt se pravděpodobně dále nevyvíjí. Členství mají možnost získat pouze členové výše uvedeného institutu.

1.3.4 ABONE

ABONE je overlay testbed, který vznikl díky sdružení Active Network. Umožňuje vývojářům služeb dynamické umisťování jejich aplikací na uzly. Dá se říci, že hlavní cíle a myšlenka je stejná jako u PlanetLabu, avšak s jedním podstatným rozdílem. ABONE je zaměřena především na rozšiřitelnost tzv. network forwarding funkcí, kdežto PlanetLab je zaměřen na konkrétní aplikaci, potažmo službu.

Kontakt a členství

Tento projekt provozuje, stejně jako XBONE, University of Southern California's Information Sciences Institute. Webové stránky http://www.isi.edu/abone/ nejsou již dostupné, takže se projekt pravděpodobně zastavil a není možné se tudíž ani stát členem.

1.3.5 DETER

DETER je výzkumný testbed. Cílem je vytvořit a provozovat nekomerční, neutrální experimentální prostředí pro výzkum cyber-security. DETER se snaží být centrem pro výměnu vědeckých informací a pro spolupráci lidí zabývajících se bezpečností a vytvářením testbeds. Obsahuje zhruba 100 uzlů a na nich zpravidla běží experimentální aplikace; dalo by se říci aplikace s nebezpečným kódem. Její zaměření je opravdu striktně orientováno na výzkum, o čemž svědčí počet uzlů i typ aplikací, které se zde testují.

Kontakt a členství

DETER spadá taktéž pod University of Southern California's Information Sciences Institute. Informace je možné nalézt na webových stránkách http://www.isi.edu/deter/. Členství není nijak

zpoplatněno a po splnění podmínek se členem může stát v podstatě každý (opět je třeba být členem nějaké organizace, jako je např. univerzita).

1.3.6 Grid

Grid je soubor tzv. middleware, zvaný Globus, který umožňuje nasadit rozsáhlé vědecké aplikace na distribuované množství systémových prostředků. To, co Grid odlišuje od overlay, je, že Grid se snaží o propojení několika velkých zdrojů systémových prostředků, které jsou mezi sebou propojeny špičkovou konektivitou. Overlay se naproti tomu skládá z mnohem více uzlů, na kterých běží větší množství aplikací, které ovšem nejsou tak náročné na konektivitu a systémové prostředky. Grid by se dal svým způsobem přirovnat k serverové farmě (viz. níže).

1.4 Serverové farmy

Pro srovnání ještě zmíním serverové farmy. Jako testbeds by se daly využít (a také se využívají), ale jsou zde zásadní rozdíly oproti overlay.

1.4.1 Specifikace serverové farmy

Serverová farma (známá také jako server cluster) je skupina serverů propojených sítí, které jsou umístěny v jedné lokaci. Hlavním důvodem je samozřejmě výpočetní výkon celé této sestavy, který slouží k náročným výpočtům. Zpravidla je každé běžící úloze přidělen primární a záložní server, takže i v případě výpadku nedojde k přerušení prováděné úlohy.

Serverové farmy jsou mezi sebou propojeny pomocí switchů, případně routerů a ke komunikaci se zpravidla používá gigabitový ethernet, případně vlastní a ještě rychlejší řešení (například Myrinet).

Další využití lze najít pro webhosting (web farm), případně jako alternativa k mainframe. Avšak oproti mainframe je zde stále jedna nevýhoda, kterou je menší spolehlivost. V serverových farmách, které čítají opravdu velké množství jednotlivých strojů, je selhání jednoho z nich naprosto běžné a je potřeba mít vyřešenou jeho náhradu, velmi rychlou rekonfiguraci celého clusteru a další věci, které je potřeba vzít v potaz, což zvyšuje nároky na celkový management.

Výkon je rozdělován softwarově a je typicky limitován celkovou chladící kapacitou a množstvím spotřebované energie, než výkonem samotných procesorů. Výkon serverových farem se tedy zpravidla udává v poměru k wattu.

Z výše uvedeného plyne hned několik zásadních rozdílu oproti overlay testbed.

1.4.1.1 Myrinet

Myrinet (http://www.myricom.com/myrinet) je vysokorychlostní síťovací systém navržený společností Myricom, který slouží k sesíťování clusterů (pracovní stanice, PC, servery, blade servery atd.). Myrinet má mnohem menší režii, než jiné standarty (např. Ethernet) a díky tomu poskytuje větší

propustnost, menší rušení a kratší odezvy. Myrinet může být využit jako tradiční síťovací systém, většinou jej však využívají konkrétní aplikace, které přistupují rovnou k rozhraní Myrinetu a obcházejí volání operačního systému. Rychlost Myrinetu může být až 10 Gigabitů za sekundu. Hlavní výhodou Myrinetu jsou ovšem velmi nízké odezvy, které jsou velmi důležité při použití v super počítačích.

1.4.2 Srovnání serverové farmy s overlay

Serverové farmy, stejně jako overlay testbed sítě, nabízí výpočetní výkon, který dostačuje k provádění náročného výzkumu, případně k běhu velmi náročných aplikací. Velkým rozdílem je ale geografické rozložení overlay oproti serverové farmě. Serverová farma se nachází na jednom místě a komunikace mezi jednotlivými stroji je zajištěna velmi rychlými linkami (gigabit ethernet, případně výše zmíněný Myrinet).

Overlay je naproti tomu vybudována na již existující síti, kde mohou být jednotlivé stroje rozmístěny prakticky po celém světě. Díky tomu může overlay nabídnout cennou možnost experimentovat se skutečným síťovým provozem, který je tvořen skutečnými uživateli.

1.5 Zhodnocení ostatních testbeds a overlays

I když se může zdát, že možností je hodně, a že není vůbec nijak zvlášť výhodné vybrat si právě PlanetLab, tak opak je pravdou.

Většina ostatních testbeds a overlays má nějaká omezení, která se týkají typů projektů, které je možné vyvíjet, omezení použitelných technologií a dostupných prostředků, komerční využití a jistě mnohá další.

Jak bude v následujících kapitolách popsáno, tak PlanetLab se snaží svoje uživatele omezovat co nejméně. V PlanetLabu je možné dělat prakticky téměř vše, na co si vzpomenete.

Navíc PlanetLab poskytuje největší počet uzlů a tím pádem i celkový výkon a množství prostředků pro koncového uživatele.

2 Specifikace Planetlab

PlanetLab je geograficky distribuovaná výpočetní platforma pro vývoj, zkoumání a nasazení celosvětových síťových služeb. PlanetLab se fyzicky rozkládá po celém světě na více než 300 místech ve více než 30 zemích. Každý z účastníků má k dispozici jeden či více izolovaných slices (slice je v češtině krajíc), což je část celkových zdrojů PlanetLabu, přidělených pomocí distribuované virtualizace.

K podpoře této inovativní infrastruktury přispívá princip nevázaného managementu. Tento princip odstraňuje vazbu mezi konkrétním operačním systémem běžícím na konkrétním uzlu a mezi celou škálou síťových služeb běžících na PlanetLabu. Aplikace a služby PlanetLabu běží na slice (chcete-li česky, tak na krajíci) celkové platformy: jde o sadu uzlů, na kterých daná služba nebo aplikace získá určitou část celkových systémových prostředků formou virtuálního stroje.

Systém distribuované virtualizace v PlanetLabu přináší několik novinek. Jde například o sadu virtuálních strojů, se kterými systém zachází jako s jedinou kompozitní entitou. PlanetLab izoluje jednotlivé služby a aplikace jednu od druhé, čímž pro tyto služby udržuje svým způsobem iluzi, že každá z těchto služeb a aplikací běží na samostatném stroji, případně na více strojích. Je také zapotřebí zajistit izolovanost sliverů (což jsou jednotlivé virtuální stroje na konkrétním jednom uzlu). Tohoto je možné docílit pomocí alokování a rozdělování systémových prostředků na daném uzlu, případně použitím kontextového označení jmenného prostoru daného systému. V neposlední řadě je nutné zajistit stabilitu a bezpečnost jednotlivých sliverů, které běží na daném uzlu. Obsah jednoho konkrétního sliveru (virtuálního stroje) přitom nemá žádný vliv na chod celé platformy. Je tedy naprosto irelevantní, zda na daném sliveru běží aplikace psaná v C++ nebo např. v assembleru.

Na obrázku 2.1 můžeme vidět schéma architektury jednoho konkrétního uzlu PlanetLabu. Na nejnižší úrovni každého uzlu běží monitor virtuálních strojů (v angličtině virutal machine monitor – VMM), který implementuje a izoluje jednotlivé virtuální stroje. Monitor virtuálních strojů také definuje API, které je použito k implementaci služeb.

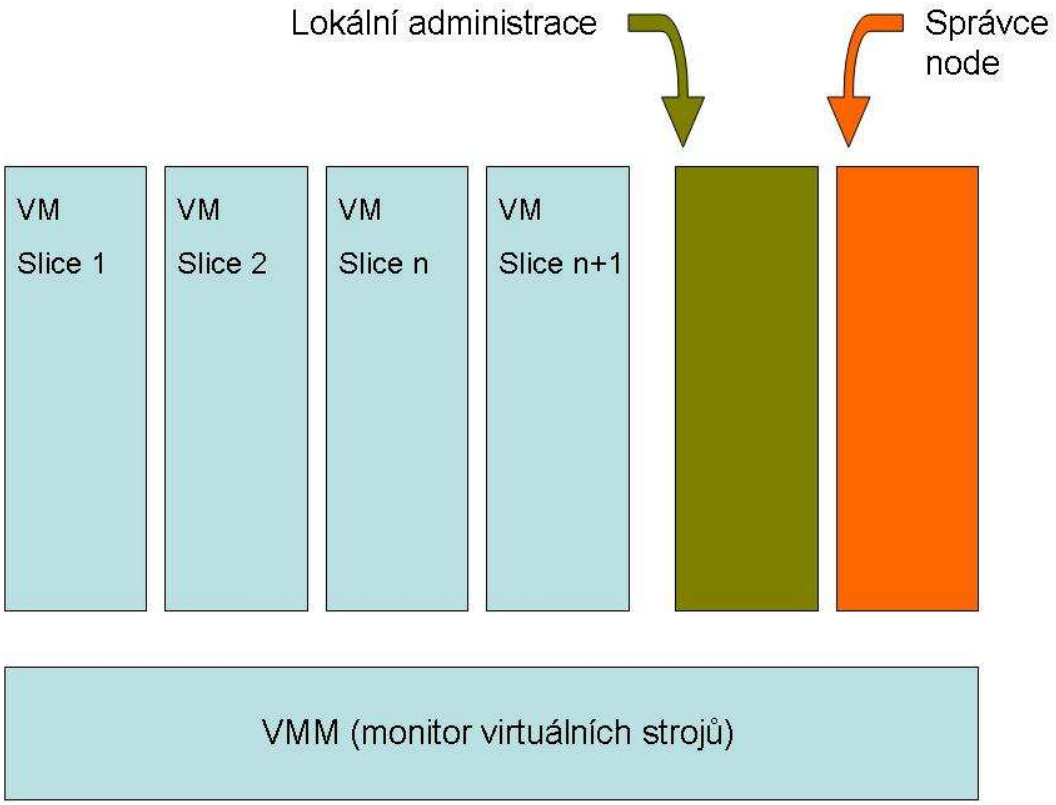
PlanetLab ve verzi 3.0 implementuje monitor virtuálních strojů jako kombinaci Linux 2.6 kernelu a sadu rozšíření kernelu. Konkrétně jde o Vservers 1.9, což je patch, který umožňuje provozování nezávislých virtuálních strojů na jednom fyzickém stroji a dále jde o SILK (Scout in Linux Kernel) modul, který umožňuje rozdělení výkonu procesoru a kontrolu sítě. Virtualizaci síťového rozhraní zajišťuje VNET.

Správce uzlu, což je privilegovaný virtualizovaný stroj běžící na monitoru virtuálních strojů, monitoruje a řídí všechny ostatní virtuální stroje běžící na daném uzlu. Obecně vzato, správce uzlu zajišťuje dodržování pravidel, která se týkají alokování systémových prostředků pro jednotlivé virtuální stroje s možností vytvoření dalšího virtuálního stroje (není nutné kontaktovat a zatěžovat monitor virtuálních strojů). Navíc jsou všechny interakce v rámci správce uzlu čistě lokální.

Jednoduše řečeno, pouze služby běžící na ostatních virtuálních strojích jednoho konkrétního uzlu jsou oprávněny kontaktovat správce uzlu, což znamená, že vzdálený přístup je vždy nepřímý a je zprostředkován pomocí služby běžící na uzlu.

Většina pravidel je hard-coded přímo ve správci uzlu, ale lokální administrátoři konkrétních uzlů mají možnost některá pravidla měnit. K tomuto účelu slouží další virtuální stroj běžící na monitoru virtuálních strojů - lokální administrace.

V PlanetLabu běží mnoho typů aplikací. Jako příklad můžeme uvést aplikace na měření výkonu sítí, multicast na úrovni aplikací, distribuované zpracovávání dotazů, distribuované hashovací tabulky, služby na alokování a přidělování systémových prostředků, služby managementu sítí, další overlay sítě a případně i další testbeds.



Obrázek 2.1: Architektura uzlu (node) PlanetLabu.

2.1 Základní terminologie PlanetLabu

Zde jsou uvedeny běžné termíny, se kterými se nevyhnutelně setkáte, pokud budete s PlanetLabem pracovat.

Tato podkapitola je částečně založena na překladu anglických materiálů na stránkách http://www.planet-lab.org/ v sekci User's Guide.

2.1.1 Site

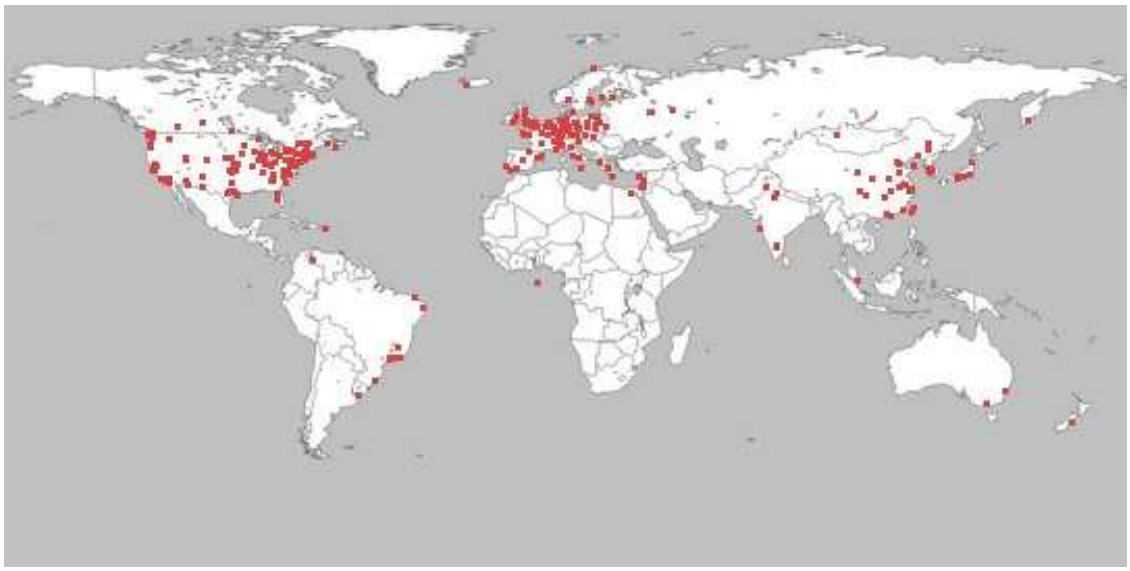
Site je lokace, na které je umístěn uzel PlanetLabu (případně více uzlů). Zkrácené jméno site je prefixem jména každého slice.

Příklad site

Jde o konkrétní místo a organizaci, takže to může být např. CESNET v Praze.

2.1.2 Uzel

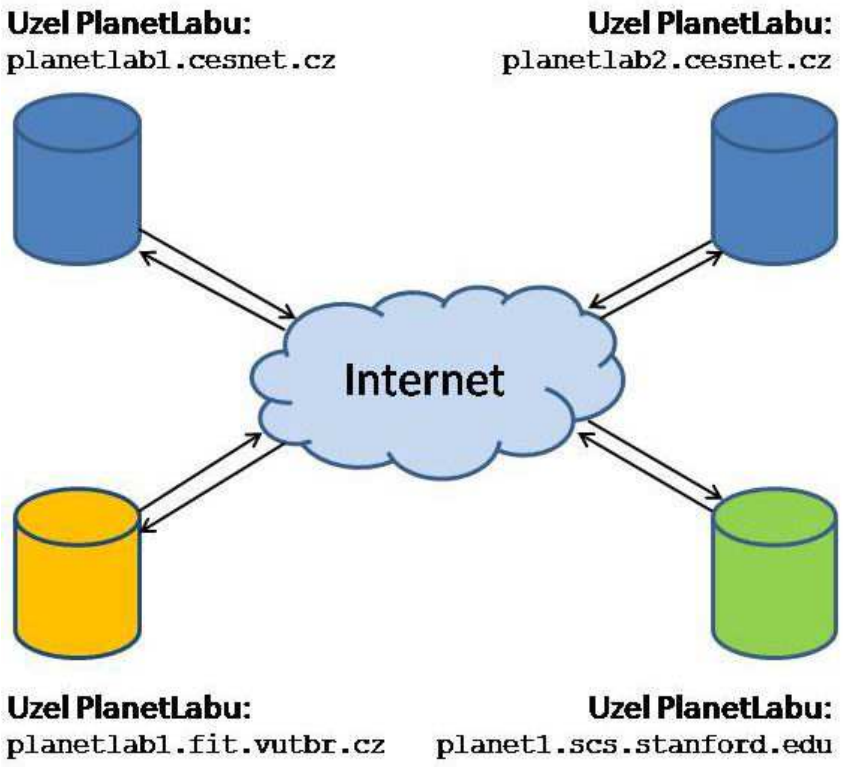
Uzel je dedikovaný server, na kterém běží komponenty služeb PlanetLabu. Uzel vždy spadá pod určitou site. Uzly PlanetLabu jsou rozmístěné doslova po celém světě (viz. obrázek 2.2) a komunikace mezi nimi je zajištěna prostřednictvím Internetu. Celkový počet uzlů PlanetLabu přesahuje číslo 800.



Obrázek 2.2: Rozmístění uzlů PlanetLabu (převzato z oficiálních stránek PlanetLabu).

Příklad uzlu

Uzel je jeden konkrétní fyzický stroj. Jako příklad můžeme uvést konkrétní stroj na Fakultě Informačních Technologií na VUT v Brně, na kterém běží software PlanetLabu. Na obrázku 2.3 jsou znázorněny jednotlivé uzly a princip komunikace mezi nimi.



Obrázek 2.3: Znázornění jednotlivých uzlů.

2.1.3 Slice

Slice je určité množství alokovaných systémových prostředků v PlanetLabu. Pro většinu uživatelů bude slice znamenat přístup k určitému množství uzlů pomocí unixového shellu. Vytváření a přidělování jednotlivých slice konkrétním uživatelům má na starosti Principal Investigator (viz. níže). Po té, co je uživateli přidělen slice, může daný uživatel přidělit k tomuto slice uzly, které má k dispozici. Až dojde k přiřazení konkrétního uzlu ke slice, proběhne vytvoření virtuálních strojů na každém přiřazeném uzlu. Slices mají omezenou časovou platnost (jejich existence je omezena v čase) a je zapotřebí je pravidelně obnovovat.

Příklad slice

Slice jsou všechny systémové prostředky, které máme v PlanetLabu k dispozici. Takže slice je, jednoduše řečeno, možnost přístupu na uzel na Fakultě Informačních Technologií na VUT v Brně, na uzel CESNETu v Praze atd.

2.1.4 Sliver

Sliver je určité množství alokovaných systémových prostředků na právě jednom konkrétním uzlu PlanetLabu.

Příklad sliveru

Jde o možnost přístupu na jeden konkrétní uzel. Např. uzel na CESNETu v Praze.

2.1.5 Virtual Server (VServer)

Každý sliver je implementován za pomoci Linux VServer, který implementuje izolaci jmenného prostoru a systémových prostředků na konkrétním fyzickém stroji (v našem případě uzlu). Síťová virtualizace je zajištěna pomocí VNET. Někdo pojmy slice, sliver a VServer občas zamění a považuje je za totožné, nicméně z výše uvedeného vyplývají jasné rozdíly zejména z hlediska architektury.

Příklad VServeru

Na každém fyzickém stroji (tj. na každém uzlu) běží právě VServer, na kterém běží jednotlivé virtuální stroje, se kterými pracují jednotlivý uživatelé (tj. slivery).

2.1.6 Principal Investigator (PI)

Na každé site musí být alespoň jeden Principal Investigator, což je osoba, která má na starost správu všech uživatelů a slices patřících k dané site. Principal Investigator je zodpovědný za chování slices, které vytvoří a přidělí uživatelům.

2.1.7 Technical Contact (Tech Contact)

Každá site musí mít alespoň jednu osobu, která se stará o instalaci, údržbu a monitorování všech uzlů patřící pod danou site. Touto osobou je Technical Contact.

2.1.8 User

User neboli uživatel je prakticky kdokoli, kdo vyvíjí a spouští aplikace v prostředí PlanetLab. Principal Investigator může být zároveň i uživatelem.

2.2 Podmínky členství a přístupu do PlanetLabu

Z hlediska právního je PlanetLab konsorcium, které je složeno především z akademických, komerčních a vládních institucí. Obyčejný člověk (rozumějme fyzická osoba) nemůže získat přístup do PlanetLabu. Přístup je možný pouze v případě, že vaše organizace je členem PlanetLabu. Aby se vaše organizace mohla stát členem, musí splnit několik důležitých podmínek, které jsou specifikované ve čtyřech následujících podkapitolách.

Text následujících podkapitol je částečně založen na překladu původních anglických materiálů na adrese http://www.planet-lab.org v příslušných sekcích. Na této adrese v příslušných sekcích také najdete podrobnější informace o této problematice.

2.2.1 Druhy členství v PlanetLabu

Pokud se organizace chce stát členem PlanetLabu, má na výběr ze čtyř možností členství:

- Privilegované členství (charter): cena tohoto členství je 300 000 USD za rok a toto členství nemá jako jediné žádné omezení.
- Plné členství (full): cena tohoto členství je 75 000 USD za rok a toto členství je omezeno možností vytvořit maximálně 10 slices.
- Přidružené členství (associate): cena tohoto členství je 25 000 USD za rok a toto členství je omezeno možností vytvořit maximálně 2 slices.
- Sponzorské členství (sponsor): cena tohoto členství je 10 000 USD za rok, u tohoto druhu členství není možné vytvořit žádný slice a tím pádem jakkoli s PlanetLabem pracovat.
- Akademické členství (academic): toto členství není nijak zpoplatněno a jeho omezením je možnost vytvořit maximálně 10 slices; toto členství je jen pro univerzity.

2.2.2 Nároky a požadavky na hostování uzlů

Pokud se chce jakákoli organizace stát členem PlanetLabu, musí vytvořit alespoň dva uzly. Vzhledem k tomu, že v sítí PlanetLab běží spousta krátkodobých experimentů a zároveň projekty, na kterých se pracuje již mnoho let, tak je nutné splnit předem stanovené podmínky, které zajistí bezproblémový a stabilní chod všech uzlů.

2.2.2.1 Odpovědnost provozovatele uzlů

- poskytnutí konektivity pro daný uzel; zajištění statické IP adresy a DNS jména (včetně reverzního záznamu)
- umístit uzel mimo firewall, což znamená jakkoli neomezovat a nefiltrovat jakýkoli provoz daného uzlu; předpokládá se oddělení uzlů PlanetLabu a izolované umístění
- umožnit týmu PlanetLabu správu daného uzlu, což znamená poskytnutí root práv a možnost instalovat a udržovat operační systém PlanetLabu; provozovatel daného uzlu root práva nemá (má však některé rozšířené pravomoce)
- zajistit odpovědnou kontaktní osobu, kterou může tým PlanetLabu kdykoli kontaktovat
- zajistit zpětnou vazbu od externích administrátorů
- zajišťovat dodržování PlanetLab AUP (Acceptable Use Policy); PlanetLab v tomto plně spoléhá na provozovatele daného uzlu

2.2.2.2 Odpovědnost PlanetLabu

- omezení celkového rozsahu přenosového pásma; administrátor provozovatele daného uzlu má plné právo omezit celkový rozsah přenosového pásma (jinak by daný uzel pravděpodobně využil veškerou dostupnou konektivitu)
- možnost filtrovat pakety z/do dané destinace; k tomuto by mělo dojít, pouze pokud je lokální administrátor daného uzlu přesvědčen, že došlo k útoku na jeho síť prostřednictvím uzlu PlanetLabu
- možnost zamezit spoofing IP adres; možnost zakázat nebezpečné pakety (např. ping of death)
- možnost limitovat množství probe paketů, případně dalších paketů, které mohou narušit fungování sítě

2.2.2.3 Hardwarové nároky

Minimálně dva uzly na každé site (tj. dva fyzické stroje) musí v současné době splňovat následující požadavky:

- CPU: Pentium 3.2 GHz, AMD 3200+ nebo ekvivalentní
- RAM: 4 GByte
- HDD: 320 GByte

Výše uvedené nároky nejsou nijak drastické a svým způsobem stačí dnešní průměrné PC.

2.2.3 Podmínky užívaní AUP (Acceptable Use Policy)

Hlavním a základním pravidlem je, že PlanetLab a veškeré jeho zdroje a prostředky nesmí být využity pro jakékoli nelegální nebo komerční aktivity. PlanetLab slouží pouze k vědeckým a vzdělávacím účelům.

2.2.3.1 Pravidla pro používání uzlů

- používání existujících bezpečnostních mechanismů (např. přístup k uzlům PlanetLabu je povolený pouze prostřednictvím SSH)
- není dovoleno jakýmkoli způsobem obcházet mechanismy auditu (to znamená, že každý uživatel musí patřit pod nějaký slice a mít vytvořený svůj účet); není možné svoji identitu při práci s PlanetLabem jakkoli skrývat, případně se vydávat za někoho jiného
- jakýmkoli způsobem hackovat uzly PlanetLabu (včetně tzv. red team experimentů)
- nepoužívat spin-wait po delší časovou periodu, nejlépe nepoužívat vůbec

2.2.3.2 Pravidla pro používání sítě

- zákaz používaní slice k získání jakýchkoli dalších systémových prostředků, které nejsou standardně k dispozici již po vytvoření slice

- zákaz používání jednoho nebo více uzlů k přetížení (flooding) sítě z hlediska dostupné konektivity na dané site
- zákaz jakéhokoli (systematický či náhodný) skenování portů či adres; zákaz sniffingu a spoofingu jakéhokoli síťového provozu

2.2.3.3 Následky porušení těchto pravidel

Porušení těchto pravidel může vést k pozastavení slice, zrušení slice, případně zrušení celé site (což je vlastně vyloučení z PlanetLabu). Některé prohřešky mohou být řešeny s příslušnými právními autoritami.

2.2.4 Konsorcium PlanetLab a jeho cíle

PlanetLab je otevřená, globálně distribuovaná platforma pro vývoj, nasazení a používání celosvětových síťových služeb. PlanetLab podporuje jak jednorázové experimenty, tak dlouhodobé a rozsáhlé projekty. Tím nejzazším a hlavním cílem je vyvíjet a demonstrovat celou novou škálu síťových služeb v celosvětovém měřítku.

Konsorcium PlanetLab bylo založeno právě pro podporu a rozvoj platformy PlanetLab v cestě za výše uvedenými cíli. Dohlíží především na rozvoj hardwarové infrastruktury celé sítě, dále na vývoj veškerého nezbytného software pro provoz, poskytuje podporu a v neposlední řadě definuje politiky a pravidla, dle kterých se celá platforma PlanetLab řídí.

2.3 Připojení a práce na PlanetLabu

Abychom mohli v prostředí PlanetLab začít pracovat, je nejprve zapotřebí vytvořit si účet. Toto můžeme učinit na webových stránkách http://www.planet-lab.org/ v sekci Account Registration (vytvoření účtu). Je potřeba být členem PlanetLabu, jinak není možné založit účet a jakkoli s PlanetLabem pracovat.

Tato podkapitola je částečně založena na překladu anglických materiálů na stránkách http://www.planet-lab.org/ v sekci User's Guide. V této sekci je možné nalézt další informace.

2.3.1 SSH klíč

K získání přístupu na jednotlivé uzly (i ty, které jsou vámi provozované) je potřeba nejdříve vytvořit pár SSH klíčů, které slouží k autentizaci.

Vzdálený přístup k jednotlivým uzlům PlanetLabu je realizován výhradně pomocí SSH s použitím RSA autentizace. RSA autentizace je založená na kryptografii veřejného klíče. Šifrování a dešifrování je zajištěno separovanými klíči, tudíž není možné získat dešifrovací klíč z šifrovacího

klíče. Vytvoření těchto klíčů je možné na jakémkoli systému UNIX (musí splňovat standard OpenSSH).

Je potřeba pečlivě chránit privátní část klíče a dodržovat zásady bezpečnosti, protože v případě zneužití klíče ponese zodpovědnost také PI dané site, případně celá organizace.

Příklad vytvoření SSH klíče:

- `ssh-keygen -t rsa -f ~/.ssh/id_rsa`

2.3.2 Vytváření slice

Pro vytvoření slice pro uživatele, případně pro přiřazení uživatele k již existujícímu slice je potřeba kontaktovat PI. Jméno slice začíná zkrácenou verzí jména site (např. cesnet_vutbr2). Po vytvoření slice a po té, co jste s ní jako uživatel asociován, je možné k této slice přiřadit konkrétní uzly. Po přiřazení uzlů ke slice může zhruba hodinu trvat, než se slice vytvoří na všech uzlech a dále než se rozšíří SSH klíč. Po té je možné se připojit.

2.3.2.1 Možné problémy a komplikace

Jednotlivé uzly jsou ve výsledku jen obyčejné osobní počítače a tak samozřejmě může dojít k jejich vyřazení (nejčastěji jde o hardwarovou závadu nebo ohrožení bezpečnosti). Ať už je daný uzel vyřazen z provozu dočasně nebo trvale, není možné se k němu připojit. Pokud uzel neodděláte ze seznamu svých uzlů (uzel přiřazený k vašemu slice) a pokud dojde ke znovu zprovoznění daného uzlu, je možné se opět připojit. Je ovšem potřeba počítat s tím, že o data uložená na uzlu pravděpodobně přijdete.

2.3.2.2 Omezení slice

Každý slice je vytvořen na požádání a jeho platnost je omezená na dva měsíce. Po vypršení této doby dojde k automatickému zrušení slice. Je třeba si uvědomit, že dojde ke zrušení slice na všech uzlech. Logickým důsledkem je ztráta všech dat.

Platnost slice je možné obnovovat a počet těchto obnovení není nijak limitován, není ovšem možné platnost prodloužit na více jak dva měsíce.

Slice je také limitován přidělenými zdroji. Jde o diskový prostor, množství paměti, počet file deskriptorů a vytížení sítě. Omezení je řešeno na bázi jednotlivých uzlů a jednotlivých slices.

2.3.3 Připojení a práce se slice

Na každém slice je po vytvoření nainstalována distribuce Linuxu Fedora Core. Samotné připojení je možné za pomoci SSH a je úplně stejně jednoduché, jako připojení k jakémukoli jinému stroji, na kterém běží SSH. Uživatel je přihlášen pod uživatelským jménem, které je shodné se jménem slice. Jelikož má každý uživatel k dispozici svoje vlastní izolované virtuální prostředí (tj. slice), může se

přihlásit i jako root a instalovat nové aplikace, vytvářet nové uživatele, kontrolovat služby a provádět další akce, které vyžadují mít root práva.

Některé operace (zejména konfigurace hardware a sítě) nejsou povoleny ani v případě, když se přihlásíme jako root. Některá rozhraní, jako např. raw sockety, je možné použít, ale je potřeba počítat s omezeními, která jsou dány použitím VNET (viz. kapitola 3.2).

Další operace, jako je otevření opravdových raw socketů, případně přístup na souborový systém jiného slice, je možné za použití PlanetLab API.

Každý slice má omezené zdroje systémových prostředků a operace související s administrativou a samotným chodem jsou pro běžný slice zakázány. I tak se ovšem může stát, že daný slice vyřadí z provozu uzel, site, případně část celé sítě. Může k tomu dojít např. neúměrným síťovým provozem, spuštěním IDS (Intrusion Detection System) nebo přilákáním pozornosti hackera či úřadů. Je tedy nezbytně nutné uvědomit si velkou míru zodpovědnosti, kterou každý uživatel pracující v síti PlanetLab má. Každý uživatel si musí hlídat svůj SSH klíč (nejlépe ho často měnit) a monitorovat přístup ke svému slice.

Příklad připojení pomocí SSH:

- `ssh -l princeton_test1 -i ~/.ssh/id_rsa planetlab-1.cs.princeton.edu`

2.3.4 Konfigurace a instalace programů na slice

Vzhledem k faktu, že se jedná o běžnou distribuci Linuxu, tak veškerá administrace a instalace je totožná jako na jakémkoli jiném operačním systému Linux. Pokud např. chybí Java, tak ji stačí prostě doinstalovat. Standardně můžeme použít balíčkovací systém RPM nebo repozitář YUM.

Příklad instalace programu pomocí YUM:

- `yum install emacs`

2.3.5 Vytvoření a nasazení aplikace

Aplikaci je možné vytvářet přímo v PlanetLabu, případně je aplikaci možno vyvíjet, vyzkoušet a otestovat na jiném libovolném operačním systému Linux a na PlanetLab danou aplikaci pouze nasadit např. pomocí SCP.

Příklad nasazení aplikace pomocí SCP:

- `scp -i ~/.ssh/id_rsa -r test1 princeton_test1@planetlab-1.cs.princeton.edu:`

2.4 Historie PlanetLabu

Kompletní historii vývoje PlanetLabu lze nalézt na adrese http://www.planet-lab.org v sekci historie. Zde je uvedena pouze stručná historie vývoje:

- Březen 2002: Larry Peterson (Princeton) a David Culler (Univerzita Berkeley a Intel Research) organizují setkání, které je zaměřeno na celosvětové síťové služby a dávají vzniknout koncepci PlanetLabu jako komunitnímu testbed, který má čítat 100 uzlů.
- Říjen 2002: Nasazení 100 uzlů na 42 místech je hotovo. Zároveň vzniká software (jde o operační systém) PlanetLab ve verzi 1.0.
- Leden 2004: PlanetLab software se dostává do verze 2.0. PlanetLab běží na více než 300 uzlech.
- Leden 2005: PlanetLab software se dostává do verze 3.0. PlanetLab běží na více než 500 uzlech.
- Leden 2007: PlanetLab software se dostává do verze 4.0. PlanetLab běží na více než 700 uzlech.

3 Technologie použité v PlanetLabu

Jádro PlanetLabu (PlanetLab Kernel) se skládá z následujících 4 položek, které dohromady dávají PlanetLab development systém (jde o operační systém PlanetLabu):

- Linux Kernel
- Linux VServer
- VNET
- CKRM

3.1 Linux Kernel

Dřívější verze PlanetLabu byly postaveny na verzi kernelu 2.4. Toto jádro bylo také použito v Redhat 9, který byl použit v dřívějších verzích jako operační systém na každém slice. S novou verzí PlanetLabu (konkrétně 3.0) došlo k nahrazení Redhat 9 operačním systémem Fedora Core, který je postaven na verzi kernelu 2.6. To je také jeden z hlavních důvodů, proč došlo ke změně kernelu na verzi 2.6 i v PlanetLabu.

Zdrojové kódy jádra PlanetLabu jsou k dispozici volně ke stažení na adrese http://cvs.planet-lab.org.

3.2 Linux VServer

Virtualizace je framework nebo metodika sloužící k rozdělení systémových prostředků počítače na více výpočetních prostředí. Linux VServer umožňuje vytvořit za pomoci virtualizace na jednom fyzickém stroji několik virtuálních strojů.

3.2.1 Specifikace Linux VServer

Varianta implementace virtualizace v Linux VServer patří mezi nejjednodušší způsoby a vzdáleně se podobá BSD JAIL (http://docs.freebsd.org/44doc/papers/jail/jail.html). Linux VServer využívá a rozšiřuje bezpečnostní systém Linuxu. Umožňuje vytvářet a spouštět izolované deamony v user-space prostředí. K implementaci je použit VPS (Virtual Private Server). Jednotlivé VPS jsou oddělené pomocí rozšířené funkce chroot a taktéž jsou oddělené procesy (PIDs, komunikace mezi procesy). Každý VPS má vlastní síťové rozhraní.

VServer umožňuje nastavit omezení pro jednotlivé VPS. Jde např. o množství paměti, množství výkonu CPU a počet file deskriptorů. Všechny VPS musí používat stejné jádro (kernel). To přináší značná omezení, ale je možné na každém VPS používat jinou distribuci Linuxu (se stejným

jádrem). To je jeden z hlavních důvodů, proč všechny slices obsahují stejnou distribuci operačního systému Linux (Fedora Core).

Princip virtualizace použitý u VServer vychází z paravirtualizace.

3.2.1.1 Paravirtualizace

Paravirtualizace je virtualizační technika představující softwarové rozhraní k virtuálním strojům, které je podobné, ale nikoli stejné, jako hardware ležící v nižší vrstvě. Přístup k hardware je v prostředí zpravidla zajišťován vrstvou virtuálního monitoru (Virtual Machine Monitor, VMM). Nad touto vrstvou jsou pak vytvářeny jednotlivé virtuální stroje (Virtual Machines, VM). Oproti plné virtualizaci má paravirtualizace menší režii, vyžaduje však určité modifikace operačního systému.

3.2.2 Historie

Projekt Linux VServer byl založen Jacquesem Gélinasem a nyní jej vyvíjí skupina programátorů vedená Herbertem Pötzlem pod licencí GPL.

3.3 VNET

VNET je modul PlanetLabu, který poskytuje virtualizovaný přístup k síti na všech uzlech PlanetLabu. VNET zároveň nahrazuje (ale je i zpětně kompatibilní) s rozhraním safe raw sockets, které bylo použito v PlanetLabu dříve. VNET poskytuje omezenou formu raw IP a raw sockets, zajišťuje izolaci síťového provozu mezi jednotlivými slices a podporuje rozhraní pro přístup k proxy IP adresám. Velmi důležité ale je, že VNET zajišťuje kompatibilitu se standardním Linux/BSD socket API.

VNET by měl být pro koncového uživatele PlanetLabu naprosto transparentní, takže valná většina uživatelů není nucená jakkoli upravovat svůj zdrojový kód, používat nějaké nestandardní programy či API. Běžný uživatel o VNET prakticky nemusí vůbec vědět, případně se obávat dopadů na funkčnost jeho programu. Jistá omezení ale samozřejmě existují.

Následující podkapitoly jsou částečně založeny na překladu původního anglického materiálu na adrese http://www.planet-lab.org/doc/vnet. V těchto materiálech je možné nalézt další podrobnější informace.

3.3.1 Sledování spojení (connection tracking)

Fungování VNET je založeno právě na sledování spojení. VNET využívá systém Netfilter (součást Linuxu), který monitoruje každý příchozí a odchozí paket a tyto pakety asociuje s jednotlivými spojeními. Tento systém umožňuje zajistit, že každý slice přijímá a vysílá pakety na spojeních, které vlastní.

Jednoduše řečeno, slice může odesílat pakety, které jsou asociované s nově vytvářenými připojeními anebo s připojeními, která už jsou vytvořena. Dále může daný slice přijímat pouze pakety, které jsou asociované nebo přiřazené k již vytvořeným připojením.

3.3.1.1 Odesílání paketu

Při odeslání IP paketu prostřednictvím socketu daný paket prochází přes VNET a tento paket je asociován s novým, případně již existujícím připojením. Pokud připojení není doposud přiřazeno k žádnému slice, tak VNET přepošle daný paket dál a přiřadí toto připojení k danému slice. Pokud je připojení již přiřazeno, ale nepatří k danému slice, tak je daný paket zahozen a aplikaci, která tento paket odeslala, se vrátí standardní chybová hláška.

3.3.1.2 Přijímání paketu

Každý příchozí paket nejdřív prochází skrz VNET a je asociován s novým nebo již existujícím připojením. Pokud je tento paket očekáván (což znamená, že připojení bylo iniciováno nebo přiřazeno některému slice), tak VNET propustí daný paket. V opačném případě je možné daný paket přijmout pouze na root slice.

3.3.2 Typy spojení

Typy spojení jsou definovány a omezeny na úrovni protokolů. VNET v současné době podporuje následující protokoly:

- TCP
- UDP
- ICMP
- GRE/PPTP

Asociování paketů s připojeními není vždy snadné. Např. některé chyby ICMP jsou asociovány s připojeními, které tyto chyby vyvolaly, než s ICMP spojením jako takovým (ICMP pakety zpravidla končí na root slice). Z toho plyne, že každý slice může obdržet zpět ICMP chyby, které zapříčinil.

3.3.3 Rezervování spojení

Jednotlivá připojení mohou být rezervována použitím systémového volání `bind()`. Jakmile je určitý port přiřazen k některému slice, tak žádný jiné slice nemůže skrz tento port komunikovat. Pokud použije slice volání `bind()` na port, který už je přiřazen jinému slice, tak je vrácena standardní chyba o nemožnosti použít tento port.

3.3.4 Omezení VNET

VNET má implementováno několik omezení, se kterými je třeba počítat (dojde k zahození paketu):

- paket není možné sledovat; zejména jde-li o multicast pakety
- spojení, se kterým je paket asociován, patří k jinému slice
- jde o ICPM paket obsahující chybu
- zdrojový port paketu je menší než 1024; tyto porty jsou vyhrazeny pouze pro root slice
- není možné jakkoli sledovat provoz na daném uzlu; uživatel může monitorovat a analyzovat provoz pouze na svém slice

Další omezení a i případná rozšíření je možné nalézt v kompletní dokumentaci k VNET.

3.4 CKRM (Class-based Kernel Resource Manager)

CKRM je framework, který zajišťuje přidělování a omezování zdrojů pro jednotlivé slices. Bližší informace a případné stažení zdrojových souborů je možné na webových stránkách http://ckrm.sourceforge.net/.

3.5 PlanetLab API

PlanetLab API není přímo součástí jádra, ale taktéž se jedná o velmi důležitý prvek celého systému.

Obsluhovat a konfigurovat PlanetLab je možné prostřednictvím jeho webových stránek. V některých případech se ovšem tato možnost nezdála jako zcela ideální a proto bylo vytvořeno API.

Uživatelé mohou pomocí API pracovat se svým slice. Jde o přiřazování uzlů k danému slice, případně o vytváření nového slice (obyčejný uživatel ovšem nemá práva vytvářet slice) a další podobné činnosti.

API samo o sobě je programové (je založeno na XMLRPC) a přístupné z příkazové řádky PlanetLabu (konkrétně jde o PlanetLab shell – plcsh). Velkou nevýhodou je, že PlanetLab shell není standardně na uzlech nainstalován. Na každém uzlu, který je přiřazený ke slice, je zapotřebí tento shell nainstalovat z CVS repozitářů PlanetLabu. Samotný shell používá syntaxi pythonu.

Pro využívání API však není vyloženě nutné používat PlanetLab shell, stačí jakýkoli skriptovací jazyk s podporou XMLRPC.

Kompletní dokumentace k PlanetLab API je k dispozici na webových stránkách http://www.planet-lab.org/doc/plc_api.

4 Vytváření aplikací a užívání PlanetLabu

Vytváření aplikací a vlastně celkové užívání PlanetLabu je jistě to nejzajímavější a nejdůležitější, co může běžného uživatele PlanetLabu zajímat. V předchozích kapitolách toho bylo poměrně hodně napsáno o technických specifikacích, různých omezeních a pravidlech, principech fungování celého systému a řadě dalších důležitých věcí.

Po přečtení toho všeho může člověk nabýt dojmu, že je to vše složité, nepochopitelné a že bude těžké s PlanetLabem pracovat.

Musím říct, že takové obavy nejsou na místě. K PlanetLabu je nejlepší přistupovat jako k běžnému UNIXovému operačnímu systému se všemi jeho výhodami a nevýhodami. Připojíte se pomocí SSH a začnete dělat svoji práci. Svým způsobem není vůbec nutné znát principy fungování celého systému. Každý uživatel dostane svůj podíl distribuované virtualizace ve formě SSH přístupu k omezenému počtu uzlů. Nic víc není potřeba z pohledu koncového uživatele řešit.

Jediným nutným předpokladem pro možnost práce s PlanetLabem je tudíž znalost jakéhokoli UNIXového operačního systému (nejlépe Linuxu).

K pochopení této problematiky slouží níže uvedená Příloha 1, ve které lze nalézt návod jak spustit velmi jednoduchou aplikaci v PlanetLabu.

5 Případová studie využití PlanetLabu

Prostředí PlanetLab nabízí velké množství příležitostí k využití. Bylo třeba vybrat jednoduchou případovou studii, která by demonstrovala možnosti PlanetLabu. Při převádění jednotlivých nápadů do reality jsem se setkal s několika úskalími, která znemožnila implementaci některých návrhů.

Je potřeba podotknout, že plně využít PlanetLab by vyžadovalo tým několika lidí, kteří budou delší dobu pracovat na nějakém projektu, který využívá PlanetLab a jeho možnosti.

5.1 Shell-over-shell

Prostředí PlanetLabu je jistě velmi zajímavé a proto bylo navrženo implementovat aplikaci, která by umožnila po ověření omezený přístup více lidem (nejpravděpodobněji studentům) do PlanetLabu za použití jednoho uživatelského účtu PlanetLabu.

Přihlašování do PlanetLabu probíhá za pomocí SSH (viz. kapitola 2) a pro všechny uživatele daného slice slouží jako jméno uživatele název slice. Toto uživatelské jméno je pro všechny uživatele stejné. Bylo tedy navrženo vytvořit aplikaci, která by umožňovala externí ověření uživatele a následné připojení do PlanetLabu s použitím výše zmíněného uživatelského jména.

Daný požadavek by byl technicky řešitelný, ale podmínky užívání PlanetLabu vyžadují, aby každý uživatel pracující s PlanetLabem měl vlastní účet vytvořený na PlanetLabu a používal vlastní RSA klíč.

Tento problém se ukázal jako neřešitelný, protože by docházelo k přihlášení více uživatelů pomocí jednoho RSA klíče, což znamená za použití účtu, který slouží právě pro jednoho uživatele. Tím by docházelo k porušení podmínek používání (AUP).

Jako řešení se nabízí vytvořit pro všechny uživatele (např. všechny studenty) účty přímo na PlanetLabu. Toto ovšem není možné, protože právo vytvářet tyto účty má pouze Principal Investigator, který by takové množství účtů téměř jistě nepovolil (bezpečnostní rizika). Pokud by přece jen došlo k vytvoření takového množství účtů, tak by bylo nutné nějakým způsobem omezit práva uživatelů, protože každý uživatel daného slice má možnost přihlásit se jako root.

5.2 Monitorování síťového provozu na uzlu

Bylo navrženo sledovat a analyzovat síťový provoz na konkrétním uzlu. Analýza síťového provozu na úrovni použitých protokolů měla poskytnout cenné statistické informace, týkající se tak rozsáhlého projektu, jako je PlanetLab. Jednalo by se zejména o poměr veškerých aplikačních protokolů oproti nezbytným řídícím protokolům a dále procentuální zastoupení jednotlivých protokolů.

Realizaci této studie zamezily dvě okolnosti vyplývající ze specifikací PlanetLabu. Tyto okolnosti nešlo jakýmkoli způsobem obejít nebo ignorovat.

První problém vyplývá z technického řešení PlanetLabu (konkrétně se jedná o VNET, který je popsaný v podkapitole 3.2). Uživatel může zpracovávat a následně analyzovat pouze pakety z jeho slice, respektive sliveru. Pakety z ostatních slices, respektive sliverů, nejsou dostupné. Ke zpracovávání a analyzování paketů by muselo docházet na root slice daného uzlu, ke kterému, ovšem, jako uživatelé nemáme přístup.

Druhým problémem by bylo porušení podmínek PlanetLabu a dle mého názoru i etiky a morálky Internetu. Kdyby byl možný přístup na root slice daného uzlu a tím pádem k veškerému síťovému provozu všech slices, respektive sliverů, tak by bylo možné z paketů určených pro jiné slices, respektive slivery, získat informace, ke kterým by se měl dostat jen uživatel daného slice, respektive sliveru.

5.2.1 Prostředky pro realizaci

K odchytávání paketů byl použit TCPDUMP, který používá pcap knihovnu. Pakety je možné odchytit, ale pouze některé. Celý problém spočívá v použití VNET (viz. podkapitola 3.2). Síťová karta na každém slice je virtuálni. Má stejnou IP adresu jako fyzická síťová karta na daném uzlu, ale to právě díky VNETu (proxied IP address). VNET se stará o naprosté oddělení jednotlivých slice (tzn. i root slice). Pakety, které odesíláme se svého slice (konkrétně z virtuálního stroje, který běží na daném uzlu), můžeme odchytit pomocí TCPDUMP. Odchytit také můžeme pakety, které jsou pro náš slice určeny.

5.3 Multicast

Vzhledem k nemožnosti používat na uzlech PlanetLabu klasický multicast přes multicastovou skupinu využívající IP adres třídy D (omezení VNET, které není možné jakýmkoli způsobem obejít), tak jsem se rozhodl analyzovat ostatní varianty multicastu a některou z nich implementovat. Samotnou analýzou, návrhem a implementací zvoleného řešení se zabývá kapitola 6.

6 Návrh a implementace multicastu

Jednotlivé uzly, respektive slice, které běží na uzlech PlanetLabu, mezi sebou žádným způsobem nekomunikují. Nemají k tomu také žádný důvod, protože monitoring jednotlivých uzlů je ošetřen interními mechanismy PlanetLabu a je realizován centrálně. Pro koncového uživatele PlanetLabu je slice na každém uzlu dostupný jako obyčejný stroj v Internetu s nainstalovaným operačním systémem Linux (konkrétně Fedora Core). Na všech takto dostupných strojích může uživatel vyvíjet a používat svoje distribuované aplikace.

Komunikaci jednotlivých uzlů (a na nich případně běžících aplikací) jsem se rozhodl zajistit právě pomocí multicastu. Srovnám zde běžný IP multicast využívající k šíření zpráv IP adresy skupiny D (použití této varianty není v PlanetLabu možné) a atomický multicast (přenos mezi jednotlivými členy multicastové skupiny může být realizován jako unicast; tato varianta je použitelná v prostředí PlanetLab).

Rozhodl jsem se tedy zanalyzovat možnosti a varianty atomického multicastu a některou z nich implementovat.

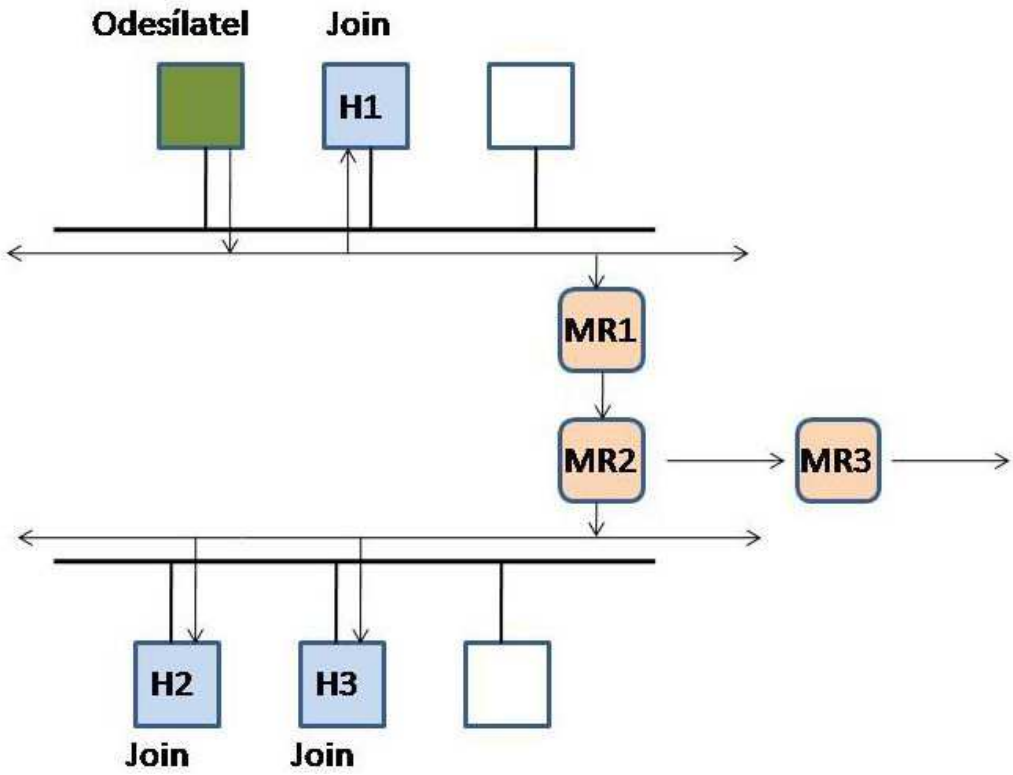
6.1 IP multicast

Jak již bylo řečeno, IP multicast využívá k šíření zpráv některou IP adresu ze skupiny IP adres D (224.0.0.0 – 239.255.255.255). Spodních 28 bitů je ID skupiny a celá adresa (32 bitů) je adresa skupiny. Tento typ multicastu funguje nad protokolem UDP a tudíž není garantované doručení zprávy a také není garantováno pořadí doručených zpráv.

Výhodou je, že zdroj odesílá UDP pakety pouze jednou, ať už je potencionálních posluchačů libovolný počet. Potencionální posluchač se přihlásí (join) do multicastové skupiny, která je identifikována již zmíněnou adresou skupiny, a z této skupiny odebírá UDP pakety. Potencionální posluchač se může ze skupiny samozřejmě také odhlásit (leave) a dále od zdroje neodebírat UDP pakety. Vzhledem k tomu, že jsou pakety odesílány pouze jednou (ať je posluchačů libovolný počet), tak tato koncepce snižuje nároky na šířku přenosového pásma celé sítě. Prostřednictvím tohoto typu multicastu jsou např. realizovány přenosy obrazu a zvuku v IPTV. Posílat unicastem každému klientovi všechny pakety by nebylo možné z důvodu neúnosných nároků na propustnost celé sítě.

Nevýhodou je naopak nutná hardwarová podpora síťových prvků (zejména routerů) a dále nedostatečný prostor multicast adres (přesněji řečeno adres pro jednotlivé skupiny).

Na obrázku 6.1 je znázorněn IP multicast, kde jedna stanice odesílá (Odesílatel) a tři stanice poslouchají (host n; Hn). Pakety prochází routery (multicast router n; MRn).



Obrázek 6.1: Znázornění běžného IP multicastu.

6.2 Atomický multicast

Informace k atomickému multicastu byly čerpány především ze zdroje [4].

Multicast některým členem skupiny je nazýván atomickým, pokud je odeslaná zpráva doručena všem členům skupiny (předpokládá se doručení všem provozuschopným a fungujícím členům) nebo žádnému členovi dané skupiny. Atomický multicast je základním předpokladem pro všechny skupinově založené síťové aktivity. Případy, které mohou vést k situacím, že některý fungující člen obdrží danou zprávu, ale jiný tuto zprávu neobdrží, je nepřípustná, protože může vyvolat nekonzistenci stavů u konkrétních členů.

Jako příklad můžeme uvést skupinu, která se zabývá vysokohorskou turistikou. Pokud je této skupině zaslána zpráva o výletu do hor, tak je zapotřebí, aby tuto zprávu obdržel každý člen.

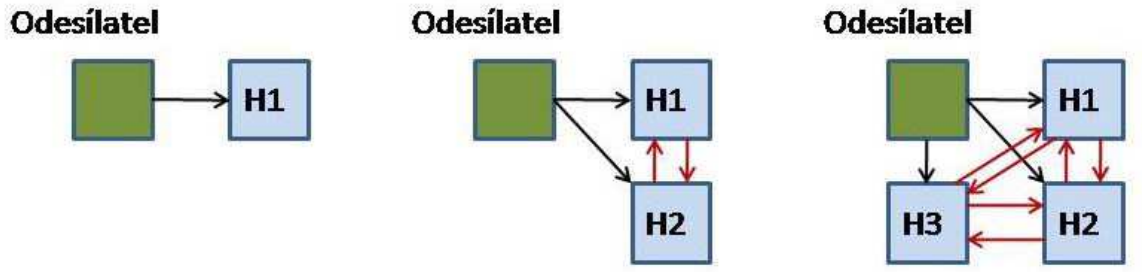
Některé systémy mají pro multicast nativní podporu. Může se jednat např. o sdílené médium jako je ethernet LAN nebo skupinu bezdrátově komunikujících zařízení, kde jsou všechna tato zařízení ve vzájemném dosahu. Pokud člen odešle skupině zprávu, tak každý další fungující člen skupiny tuto zprávu obdrží i navzdory selhání samotného odesílatele nebo dalších členů.

Existují ovšem systémy, kde takováto nativní podpora není, případně ji není možné z nějakého důvodu použít. Pak je multicast realizován jako point-to-point komunikace (unicast) a implementace takového atomického multicastu se stává netriviální.

Na následujících řádcích nastíním realizaci mulitcastu pomocí sekvence unicast komunikace. Mějme skupinu **n** členů **{0,1,2,3,……,n -1}**. Implementace takového multicastu zaručující doručení na spolehlivě realizovaných spojích je znázorněna níže.

```
Odesílatel:                    Příjemce:
i = 0;                         if m == nova zprava
while i != n                      prijmi_zpravu;
   posli_zpravu m -> i;           posli_zpravu m -> ostatni_clenove;
   i++;                        else
                                  zahod_zpravu;
```

Pokud odesílatel jakýmkoli způsobem selže během odesílání, tak každý příjemce může rozeslat všechny zprávy, které zatím od odesílatele obdržel. Toto splňuje požadavky atomického multicastu. Tento přístup má nevýhodu většího množství poslaných zpráv, která roste s komplexností $\mathbf{n^2}$. Na obrázku 6.2 je tato skutečnost názorně zobrazena.



Obrázek 6.2: Zobrazení množství posílaných zpráv.

Můžeme rozlišit dva druhy multicastu: běžný (basic) a spolehlivý (reliable). Běžný multicast může řešit selhání některého člena (ale také toto selhání vůbec nemusí nijak ošetřit). Spolehlivý multicast toto selhání musí brát v potaz a také je ošetřit.

Spolehlivý multicast by měl zajišťovat tři následující podmínky:

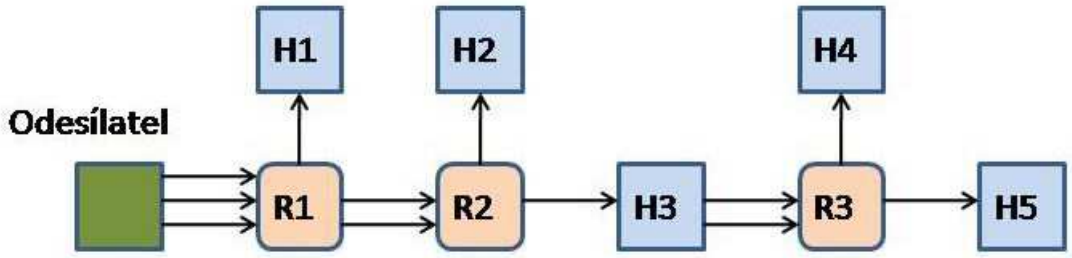
- **Validita**. Pokud člen skupiny odešle zprávu **m**, tak zpráva **m** je posléze doručena.
- **Shoda**. Pokud člen skupiny doručí zprávu **m**, tak všichni členové doručí zprávu **m**.
- **Integrita**. Každý člen skupiny doručí zprávu **m** ostatním členům maximálně jednou.

6.2.1 Multicast na úrovni aplikační vrstvy

Pokud není možné využít IP multicast (z jakéhokoli důvodu), tak se jako alternativa nabízí multicast na úrovni aplikační vrstvy. U IP multicastu jsou data nebo zprávy replikovány na routerech a tyto routery musí uchovávat informace o stavu dané skupiny, což zvyšuje jejich zátěž.

Multicast na úrovní aplikační vrstvy funguje jako overlay síť, ve které jsou data nebo zprávy replikovány jednotlivými členy dané multicastové skupiny. Implementace je formou point-to-point zpráv, které si jednotliví členové posílají mezi sebou.

Nevýhodou tohoto řešení je vyšší nárok na šířku přenosového pásma, protože identické zprávy jsou posílány mezi členy (host n; Hn) několikrát přes stejnou linku a stejný router (router n; Rn). Viz. obr. 6.3.



Obrázek 6.3: Znázornění několikanásobného zasílání zpráv.

6.2.2 Uspořádaný multicast (ordered multicast)

Informace k uspořádanému multicastu byly čerpány především ze zdroje [4].

Každá mutlicastová skupina má jiné nároky. V ideálních případech (z hlediska jednoduchosti návrhu a implementace) není nutné řešit spolehlivost ani pořadí doručovaných zpráv. Atomický multicast zajišťuje spolehlivé doručení dané zprávy, ale pořadí zasílaných zpráv, které jsou následně doručovány jednotlivým členům, žádným způsobem neřeší.

Pokud potřebujeme tento problém řešit, použijeme k tomu uspořádaný multicast (ordered multicast). Koncepcí uspořádaného multicastu existuje několik verzí. Pro atomický multicast se zdá jako nejlepší varianta úplně uspořádaný multicast (total ordered multicast).

6.2.2.1 Úplně uspořádaný multicast (total ordered multicast)

V této formě atomického multicastu musí každý člen multicastové skupiny obdržet všechny zprávy v identickém pořadí. Každý člen takovéto skupiny má svoji vlastní frontu **Q**, ve které zpracovává příchozí zprávy. Pro člena **i** (fronta **Q.i**) a pro člena **j** (fronta **Q.j**) platí **Q.i** = **Q.j**.

6.3 Srovnání IP multicastu s atomickým multicastem

Zásadním rozdílem, kterým se bude IP multicast vždy lišit od atomického multicastu realizovaného unicastem, je množství redundantních identických zpráv či dat, které jsou v rámci dané multicastové skupiny odeslány mezi jejími členy. Toto je nesporná výhoda IP multicastu, protože zde k žádnému odesílání duplicitních zpráv či dat nedochází.

Ostatní vlastnosti, jako je již zmíněná spolehlivost, pořadí doručených zpráv, ošetření různých výjimečných stavů a podobně, je možné řešit u obou těchto koncepcí.

Výhodou atomického multicastu je spolehlivost, která zaručuju doručení všech zpráv či dat všem členům ve skupině. U IP multicastu není tento problém běžně řešen, ale samozřejmě existuje velké množství multicastových skupin, ve kterých tento problém není třeba řešit. Pak je IP multicast ideální varianta (pokud jej ovšem můžeme použít).

6.4 Návrh multicastu pro PlanetLab

Rozhodl jsem se navrhnout multicast systém, který bude sloužit ke komunikaci jednotlivých uzlů PlanetLabu. Tento systém bude realizován právě pomocí atomického multicastu realizovaného unicastem. Návrh jsem rozdělil na následující podkapitoly, ve kterých je popsáno, jak danou část řeším a s jakými problémy jsem se během návrhu setkal.

6.4.1 Návrh člena skupiny

Každý člen skupiny je schopen fungovat naprosto nezávisle na ostatních členech. Skupina není nijak centrálně řízena a žádný člen neplní funkci arbitra. Díky tomu se může jakýkoli člen skupiny kdykoli odhlásit, případně výpadek daného člena neovlivní zbytek členů a fungování skupiny.

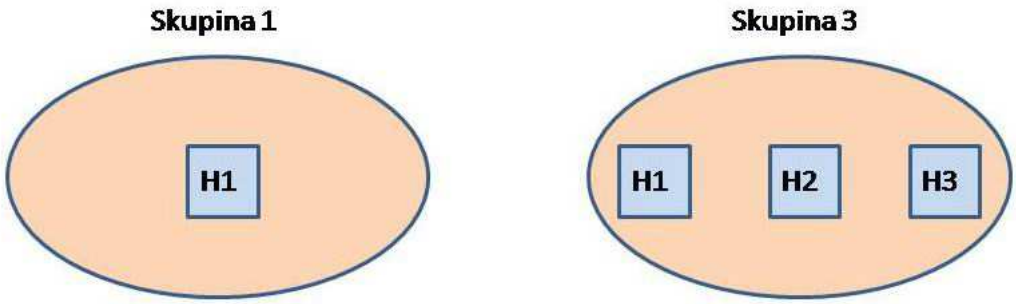
Každý člen skupiny je schopen zároveň zprávy odesílat i přijímat a každý člen si udržuje svůj vlastní seznam členů dané skupiny.

6.4.2 Návrh skupiny, přihlášení a odhlášení ze skupiny

Atomický multicast nemá nijak definovanou celou skupinu (její identifikátor), složení skupiny a případné přihlašování a odhlašování. Všechny tyto problémy bylo potřeba vyřešit.

6.4.2.1 Vytváření a identifikace skupiny

Skupinu tvoří minimálně jeden člen a skupina je identifikována taktéž minimálně jedním členem (tzn., že skupinu deseti členů identifikuje alespoň jeden její libovolný člen). Na obrázku 6.4 je možné vidět příklady skupin. Skupina 1 obsahuje člena H1. Tato skupina je identifikována právě členem H1. Skupina 3 obsahuje členy H1, H2 a H3. Tato skupina je identifikována členem H1 nebo H2 nebo H3, případně kombinací těchto členů.



Obrázek 6.4: Znázornění skupin.

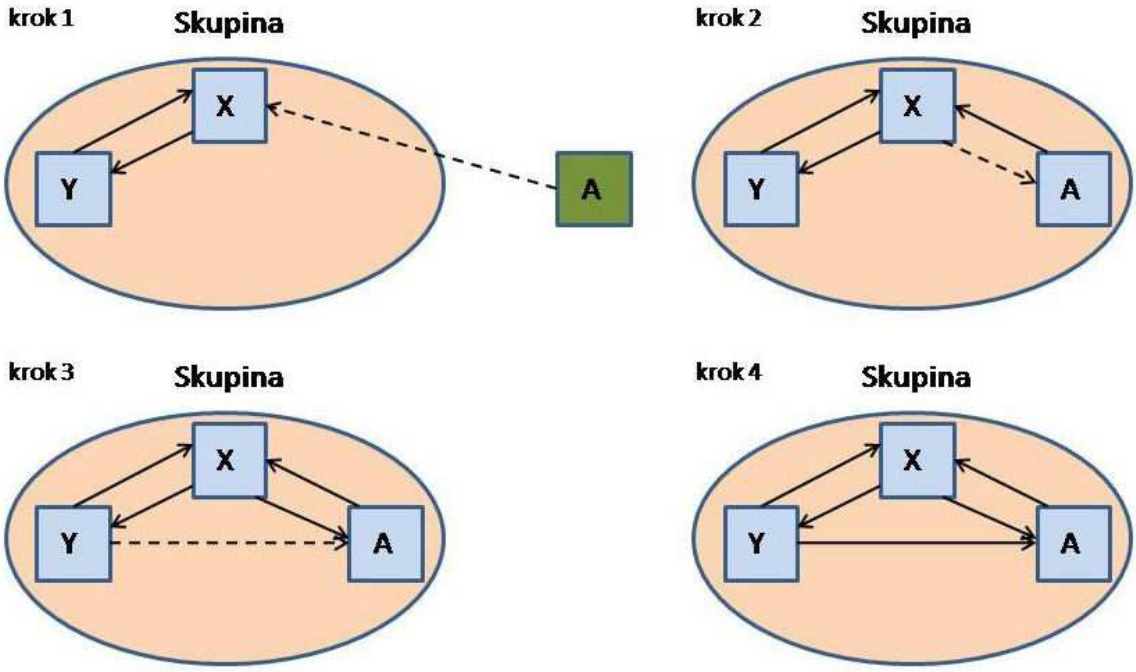
6.4.2.2 Přihlášení do skupiny

Jak již bylo řečeno, skupinu identifikuje alespoň jeden její člen. Pokud se tedy bude chtít do dané skupiny připojit např. budoucí člen A, stačí poslat zprávu alespoň jednomu jejímu členovi. Z toho plyne, že je nutné znát alespoň jednoho člena (např. člen X). Člen A se stane členem skupiny v okamžiku, kdy alespoň jeden současný člen skupiny bude o existenci člena A vědět. Přihlášení probíhá následovně (uvažujme skupinu, která se skládá z člena X a člena Y; člen A se chce přihlásit):

- člen A odešle zprávu msg_A členovi X
- člen X analyzuje zprávu msg_A a zkontroluje, zda o existenci člena A již ví; pokud ne, zapíše si člena A do seznamu členů skupiny, o kterých ví
- člen X odešle zprávu msg_A všem členům skupiny, o kterých ví; zprávu neodesílá sám sobě a členovi A, zprávu tedy odešle členovi Y
- člen Y analyzuje zprávu msg_A a zkontroluje, zda o existenci člena A již ví; pokud ne, zapíše si člena A do seznamu členů skupiny, o kterých ví
- člen Y odešle zprávu msg_A všem členům skupiny, o kterých ví; zprávu neodesílá sám sobě a členovi A, zprávu tedy odešle členovi X
- člen X analyzuje zprávu msg_A a zjistí, že tuto zprávu již obdržel; člen X už tuto zprávu nikam neodesílá

V tomto okamžiku již oba původní členové (X a Y) ví o existenci člena A. Pokud bude tedy člen X nebo Y odesílat zprávu, tak tato zpráva dojde všem členům skupiny (X, Y, A). Člen A se o existenci člena Y dozví, jakmile Y odešle libovolnou zprávu. I kdyby člen Y žádnou zprávu neodeslal, a člen A by se o existenci člena Y nedozvěděl, tak zprávy od člena A budou členovi Y doručeny prostřednictvím člena X.

Celý proces přihlášení a komunikace je znázorněn na obrázku 6.5. Nepřerušovaná šipka v obrázku vyjadřuje existenci již známé vazby mezi členy, přerušovaná šipka vyjadřuje novou vazbu mezi členy skupiny.



Obrázek 6.5: Znázornění přihlašování do skupiny.

6.4.2.3 Odhlášení ze skupiny

Odhlášení ze skupiny probíhá zasláním specifické zprávy všem členům skupiny. Členové skupiny neodstraní odhlášeného člena ze seznamu členů, o kterých ví, ale pouze přestanou tomuto členovi posílat zprávy (probíhá pomocí nastavení příznaku).

K obnovení posílání zpráv odhlášenému členovi dojde v okamžiku, kdy tento odhlášený člen opět odešle zprávu do skupiny.

6.4.2.4 Výpadek člena skupiny

Pokud člen skupiny jakýmkoli způsobem vypadne (odpojení od sítě, porucha HW, pád operačního systému) a neodhlásí se, tak se každý člen skupiny takto nedostupnému členovi pokusí doručit zprávy pětkrát. Pokud doručení selže pětkrát po sobě (např. po čtyřech neúspěšných pokusech dojde k doručení zprávy; od doručení se opět zpráva doručuje pětkrát), tak je tento člen označen za odhlášeného. Počet pokusů o doručení je možné u každého člena nastavit (probíhá pomocí nastavení příznaku).

K obnovení posílání zpráv odhlášenému členovi dojde v okamžiku, kdy tento odhlášený člen opět odešle zprávu do skupiny.

6.4.3 Odesílání zpráv

Každý člen odesílá svoje zprávy všem členům skupiny, o kterých ví. Každá zpráva obsahuje identifikaci člena, který zprávu vytvořil (originator), identifikátor zprávy, časovou značku (time stamp) a pak samotný text zprávy. Identifikátor zprávy je číslo 1 až 10 (přiděluje se opakovaně

cyklicky) a časové značka je čas ve vteřinách a mikrosekundách od 1.1.1970. Rozsah identifikátorů zprávy je možné nastavit (je možné cyklicky přidělovat např. 1 až 1000). Hodnotu rozsahu je vhodné zvolit dle parametrů sítě a typu dat, které budou posílány.

Odesílání doručených zpráv (tj. zprávy od jiných členů) probíhá až po analýze těchto zpráv v sekci přijímání zpráv. Na základě analýzy příchozí zprávy je rozhodnuto, zda bude zpráva odeslána dále všem známým členům skupiny.

Příklad zprávy: členX#5#1235641.15616#Hello there#.

6.4.4 Příjem zpráv

Každá přijatá zpráva je po obdržení analyzována. Nejdříve člen skupiny zjišťuje, zda o původci (originator) této zprávy ví. Pokud ne, tak dojde k uložení nového člena do seznamu známých členů.

Následuje kontrola samotné zprávy. Je zkontrolován identifikátor dané zprávy, zda odpovídá identifikátoru očekávané zprávy. Pokud identifikátor zprávy odpovídá, tak je zpráva zpracována a odeslána všem známým členům a je nastaven nový identifikátor očekávané zprávy.

Pokud je doručena zpráva, kde neodpovídá identifikátor zprávy identifikátoru očekávané zprávy, tak následuje kontrola časových značek. Každý člen má k dispozici časové značky předchozích 10 zpráv (opět je možné nastavit počet zpráv, ke kterým se uchovávají časové značky). Na základě kontroly časových značek je rozhodnuto, zda jde o zprávu, která již byla obdržena (tato zpráva je zahozena a není posílána dále), případně zda jde o zprávu, která ještě nebyla obdržena. V tomto případě by došlo ke ztrátě zprávy, která patří mezi poslední dvě obdržené zprávy. Člen o této události podá informaci (ke ztrátám zpráv nedochází; komunikace mezi členy je realizována prostřednictvím TCP/IP). Tato zpráva by byla následně přeposlána všem známým členům.

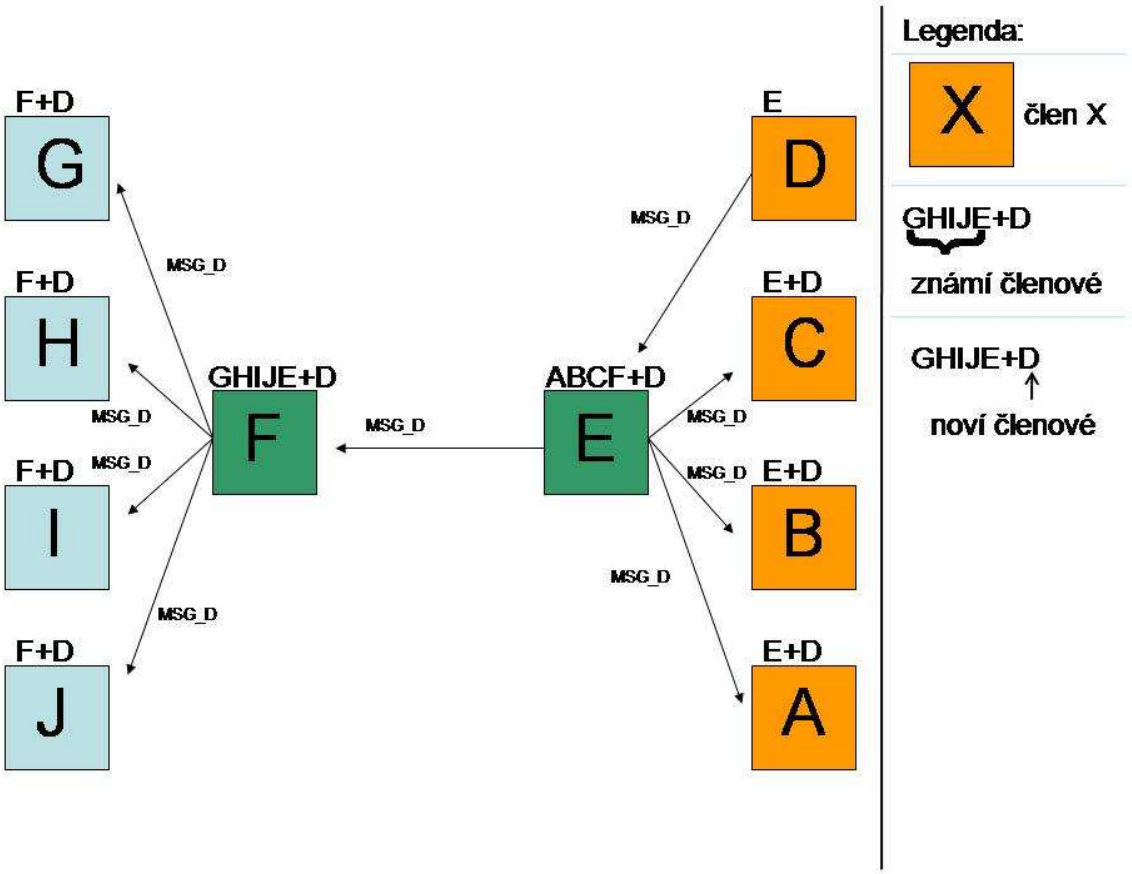
Každý člen je schopen současně přijímat a zpracovávat více zpráv najednou.

6.4.5 Demonstrace odeslání zprávy a její analýzy

Na obrázku 6.6 je možné názorně vidět, jak člen D odešle zprávu MSG_D. Zpráva se šíří mezi jednotlivými členy a nakonec je také všem členům doručena. Všichni členové se zároveň dozvědí o existenci člena D (nový člen je pro znázornění oddělen znakem “+”).

6.4.6 Členství ve více skupinách

Vzhledem k tomu, že skupinu identifikuje alespoň jeden její člen (viz. podkapitola 6.4.2), tak by tento člen nemohl být členem více skupin (protože by pak jednoznačně neidentifikoval jednu skupinu). Každý člen je v implementovaném návrhu identifikován pomocí IP adresy a portu. Členství ve více skupinách se dá tedy zajistit pomocí odlišného portu. Např. pro skupinu X by byl člen identifikován jako 210.25.32.12:15358 a pro skupinu Y by byl daný člen identifikován jako 210.25.32.12:15359. V PlanetLabu jsou pro uživatele k dispozici porty 1024 a výše, což je naprosto dostačující počet.



Obrázek 6.6: Znázornění odesílání zprávy.

6.5 Implementace navrženého řešení

V této části popíšu vlastní implementaci návrhu. Popíšu postup realizace navrženého řešení a problémy, se kterými jsem se při této realizaci setkal. Navržené řešení jsem implementoval v jazyce C a ke komunikaci jednotlivých členů je použito TCP/IP (RFC 793) a BSD socketů. Jako vývojové prostředí jsem používal převážně editor Kate z prostředí KDE. K překladu je použit kompilátor gcc. Projekt byl překládán a testován výhradně na operačním systému Linux, konkrétně na distribucích CentOS a Fedora Core.

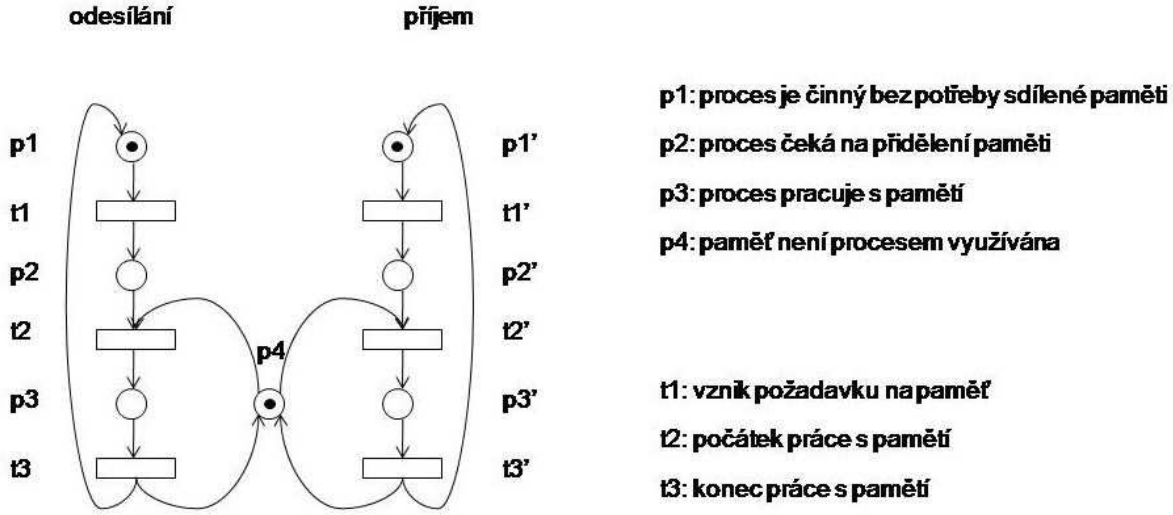
Popis implementace popíšu v následujících podkapitolách. Odesílání zpráv je řešeno v samostatném procesu. Přijímání zpráv, jejich následná analýza a případné následné odesílání je řešeno taktéž v samostatném procesu. Tyto dva jinak nezávislé procesy sdílejí informace o ostatních členech (jde o seznam všech známých členů a informací k nim).

6.5.1 Sdílené informace a funkce main

Oba procesy vyžadují sdílený přístup k informacím, které se týkají ostatních členů skupiny. Použití roury (pipe) ke komunikaci procesů se ukázala jako nevhodné, vzhledem k množství informací, které je potřeba sdílet. Jako nejvhodnější řešení se ukázalo použití sdílené paměti. Tato paměť je alokována

pomocí funkce `shmget()` ve funkci `main()`. Tato paměť je pak v případě přístupu jednotlivě přiřazována pomocí funkce `shmat()` a odebírána pomocí funkce `shmdt()`.

Přístup do paměti je řízen nepojmenovaným POSIX semaforem, který je ve funkci `main()` inicializován pomocí funkce `sem_init()`. Snižování hodnoty semaforu před přístupem do paměti je prováděno funkcí `sem_wait()` a zvýšení hodnoty semaforu je prováděno funkcí `sem_post()`. Na obrázku 6.7 je možné vidět princip přístupu do paměti (Petriho Síť).



Obrázek 6.7: Princip přístupu do sdílené paměti.

Data ve sdílené paměti jsou typu `CLIENT`. Jedná se o strukturu, která obsahuje všechny potřebné informace:

```
typedef struct
{
int msg_id; // identifikator ulozene zpravy
long int sec; //casova znacka obsahujici vteriny od 1.1.1970
long int usec; //casova znacka obsahujici zbyle mikrosekundy
}MSG;

typedef struct
{
char port[10]; //cislo portu clena
int reachability; //informace o dosazitelnosti clena
int msg_expected; //pristi ocekavana zprava
char client_id[INET_ADDRSTRLEN]; // IP adresa clena
MSG entire_msg[MSG_BUF]; // informace o poslednich 10 zpravach
}CLIENT;
```

Konstanta `MSG_BUF` určuje počet zpráv, ke kterým se budou uchovávat časové značky a zároveň určuje rozsah identifikátorů zpráv (nastaveno na 10, viz. podkapitoly 6.4.3 a 6.4.4). Všichni členové jedné skupiny musí mít stejnou hodnotu `MSG_BUF`, jinak by neprobíhala kontrola zpráv korektně.

K inicializaci informací ve sdílené paměti slouží funkce `init_clients()`. Informace o jednotlivých členech (konkrétně IP adresa a port) jsou nahrány do sdílené paměti ze souboru `ip_list` ve funkci `load_clients()`.

Funkce `get_my_ip()` slouží k získání vlastní IP adresy.

Následně ve funkci `main()` dochází k volání funkce `fork()`, která vytvoří dva nezávislé procesy, z nichž jeden má na starost odesílání zpráv a druhý přijímání zpráv.

6.5.1.1 Vstupní pamatery

Po spuštění dochází ke kontrole vstupních parametrů. Jako vstupní parametr je možné použít –p, který určí číslo portu, na kterém bude tento člen poslouchat. Standardně jde o číslo portu 15358. Parametr je tudíž volitelný.

Druhý parametr je taktéž volitelný a určuje počet pokusů o doručení zprávy (standardně 5 pokusů po sobě). Tento parametr má tvar –r.

Poslední parametr (opět volitelný) má tvar –f, který určí soubor, do kterého se budou ukládat přijaté zprávy. Pokud parametr není použit, tak se zprávy neukládají.

Parametr --help vypíše nápovědu.

6.5.1.2 Soubor ip_list

Soubor `ip_list` obsahuje na každém řádku adresu jednoho člena, případně ještě port tohoto člena. Každý řádek je ukončen znakem "#". Port je volitelný a může být uveden za adresou (oddělený dvojtečkou). V případě neuvedení portu je pro daného člena nastaven implicitní port 15358.

Příklad obsahu souboru `ip_list`:

123.123.123.123:9999#

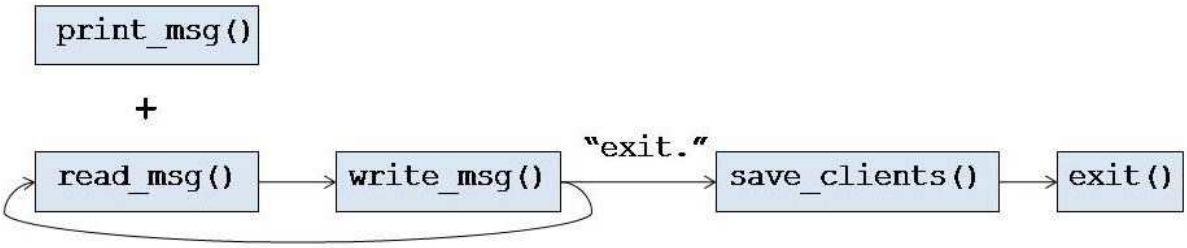
213.213.213.213#

6.5.2 Odesílání zpráv

Zpráva je načítána z klávesnice ve funkci `read_msg()`, dokud není zadán znak "." Zpráva obsahuje identifikátor (IP adresu a port) původního odesilatele (originator), dále obsahuje identifikátor zprávy a časovou značku. Po té následuje text samotné zprávy. Jednotlivé položky zprávy jsou od sebe odděleny pomocí znaku "#" (viz. podkapitola 6.5.4).

Takováto zpráva je následně odeslána pomocí funkce `write_msg()` všem známým členům, kromě původce zprávy a sama sebe. Zpráva je zobrazena pomocí funkce `print_msg()`.

Pokud je zadána z klávesnice zpráva “exit.”, tak dojde k odhlášení ze skupiny odesláním zprávy “exit.” všem známým členům. Následně dojde k uložení všech informací o klientech (členech) do souboru `ip_list` pomocí funkce `save_clients()` a pak k ukončení programu.



Obrázek 6.8: Znázornění toku dat při odesílání zprávy.

6.5.3 Příjem zpráv

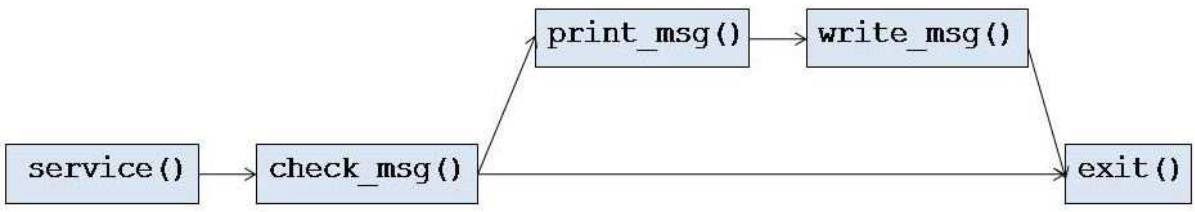
Zpráva je přijata pomocí funkce `service()` a na takto přijatou zprávu je zavolána funkce `check_msg()`. Nejdříve tato funkce zjistí, zda již o původci obdržené zprávy (originator) ví. Pokud ne, uloží nového člena do seznamu členů (použití sdílené paměti). Dále tato funkce zjistí, zda již tato zpráva nebyla obdržena a zda identifikátor zprávy odpovídá očekávanému identifikátoru. Pokud zpráva nebyla obdržena a identifikátor odpovídá, tak je zavolána funkce `print_msg()`, která tuto zprávu zobrazí. Dále je zavolána funkce `write_msg()`, která zprávu odešle všem známým členům, stejně jako v sekci odesílání zpráv.

Pokud zpráva již byla obdržena, tak dojde k zahození zprávy. Tato zpráva není nijak zobrazována ani přeposílána.

Funkce `check_msg()` ověřuje pořadí zpráv a případně i detekuje chybějící zprávu. Vzhledem k použití TCP/IP ke ztrátám zpráv nedochází.

Pokud je obdržena zpráva “exit.”, tak dojde k nastavení informace o tom, že člen je nedostupný. Informace o dostupnosti je obnovena po jakékoli další obdržené zprávě od tohoto člena, kromě zprávy “exit.”.

Zpráv je možné přijmout více najednou, protože server je neblokující.



Obrázek 6.9: Znázornění toku dat při příjmu zprávy.

6.5.4 Formát zprávy

Zpráva nejdříve obsahuje identifikaci původního odesílatele – IP adresu a port. Tyto dvě informace jsou odděleny znakem ":". Následuje znak "#" a po něm identifikátor zprávy. Dále následuje znak "#" a po něm časová značka. Vteřiny a mikrosekundy jsou od sebe odděleny znakem ".". Po té opět následuje znak "#" a pak text samotné zprávy ukončený opět znakem "#". Délka celé zprávy je omezena na 1024 znaků (při vytváření zprávy je na to třeba pamatovat!). Formát zprávy a její velikost nechávám na potřebách konkrétní aplikace (viz. podkapitola 6.8).

Příklad zprávy: 123.123.123.123:15358#5#1235641.15616#Hello there.#.

6.6 Nasazení a testování navrženého řešení

Pro testování jsem se rozhodl aplikaci nasadit na 10 uzlů PlanetLabu. Vybral jsem záměrně uzly rozmístěné po celém světě, nejen na jednom kontinentu. Více než 10 uzlů jsem nepoužil, protože data jsou zadávány ručně z klávesnice a obsluhovat přes SSH více jak 10 uzlů najednou je náročné. Zde jsou použité uzly:

- 195.113.161.82:15358#planetlab1.cesnet.cz
- 195.113.161.83:15358#planetlab2.cesnet.cz
- 12.46.129.21:15358#planet1.berkeley.intel-research.net
- 128.232.103.201:15358#planetlab1.xeno.cl.cam.ac.uk
- 128.112.139.71:15358#planetlab-1.cs.princeton.edu
- 128.112.139.74:15358#planetlab-6.cs.princeton.edu
- 141.76.45.17:15358#planet1.inf.tu-dresden.de
- 203.178.143.10:15358#planetlab1.sfc.wide.ad.jp
- 131.254.208.10:15358#peeramide.irisa.fr
- 150.254.212.147:15358#planetlab-1.man.poznan.pl

6.6.1 Nasazení aplikace

K nasazení aplikace jsem použil CoDeploy, což je jeden z nástrojů pro práci s PlanetLabem. Všechny tyto nástroje spadají pod projekt CoDeeN (viz. Příloha 2). Práce s CoDeploy je velmi jednoduchá a je popsána na webových stránkách http://codeen.cs.princeton.edu/codeploy/. Na těchto stránkách je také možné CoDeploy stáhnout. K nasazení aplikace prostřednictvím CoDeploy je potřeba soubor obsahující adresy uzlů, na které se mají požadované soubory, případně adresáře, přenést. Tento soubor je na přiloženém CD.

Pokud není možné z jakéhokoli důvodu použít CoDeploy, tak na přiloženém CD je možné nalézt jednoduchý skript, který prostřednictvím SCP postupně přenese soubory nebo adresáře na výše uvedené uzly.

Připojení na jednotlivé uzly je realizováno přes SSH. Následně je spuštěna aplikace atomického multicastu (amul) na každém uzlu. Pro přihlášení na jednotlivé uzly je možné použít deset velmi jednoduchých skriptů (k dispozici na CD).

6.6.2 Testování aplikace

Aplikace byla spuštěna na výše uvedených uzlech a tyto uzly mezi sebou komunikovaly. Na všech uzlech bylo pořadí doručených zpráv identické. Během testu vypadl uzel planetlab-6.cs.princeton.edu, což dle očekávání neovlivnilo fungování ostatních členů této multicastové skupiny. Jednotliví členové postupně ukončili komunikaci zasláním zprávy "exit.". Zde je část výpisu z log souboru, který byl pořízen na uzlu planetlab1.cesnet.cz (na CD je k dispozici celý soubor):

From: 12.46.129.21:15358
MSG(1): this is me!!!.

From: 195.113.161.83:15358
MSG(1): and me :).

From: 195.113.161.82:15358
MSG(1): hi, finish it.

From: 150.254.212.147:15358
MSG(2): exit.

From: 131.254.208.10:15358
MSG(2): exit.

From: 203.178.143.10:15358
MSG(2): exit.

Uzel (člen) planetlab1.cesnet.cz ukončil komunikaci jako poslední, proto obsahuje informace o ukončení všech ostatních členů (a nakonec sebe sama).

Doručování zpráv bylo téměř okamžité. Odezva u geograficky nejvzdálenějších uzlů odpovídala skutečnosti (zejména odezva na japonský uzel byla cca 500 milisekund). Každopádně všechny zprávy byly doručeny ve stejném pořadí všem členům. Nedošlo ke ztrátě žádné zprávy.

6.7 Zhodnocení navrženého řešení

Toto řešení bylo testováno v PlanetLabu a zároveň navrženo pro PlanetLab. Jedná se o možný způsob multicastové komunikace mezi jednotlivými uzly PlanetLabu. Využitelnost je v celé řadě aplikací,

které potřebují multicast ke své komunikaci, ale nemohou v PlanetLabu použít klasický IP multicast. Využití se samozřejmě nevztahuje jen na síť PlanetLab, tuto koncepci je možné použít v jakékoli jiné síti, kde není možné využívat IP multicastu.

Tato varianta atomického multicastu zaručuje spolehlivé doručení všech zpráv a zároveň i pořadí těchto zpráv. Dalo by se říci, že její jedinou nevýhodou je již zmiňované množství zasílaných redundantních zpráv mezi jednotlivým členy dané multicastové skupiny (vysvětleno v kapitole 6.2 a znázorněno na obrázku 6.2). Velké množství dat a zároveň velké množství členů by mohlo způsobit neúměrné zatížení sítě. Např. pro technologii IPTV je svým způsobem možné využít pouze IP multicast, kde nedohází k odesílání stejných dat vícekrát.

Je ale možné již zmiňovanou nevýhodu obrátit v užitečnou vlastnost. Budeme uvažovat použití UDP/IP (RFC 768) namísto TCP/IP a dále budeme uvažovat 10 členů multicastové skupiny. Každý člen obdrží zprávu X devětkrát, pokud nedojde ke ztrátě žádné zprávy. I v případě velmi nespolehlivých linek (např. ztráta 8 zpráv z 9) dojde k doručení této zprávy. Pokud budeme uvažovat např. 100 členů, tak může dojít ke ztrátě 98 zpráv z 99. S množstvím zaslaných duplicitních zpráv roste spolehlivost doručení zprávy, což není zanedbatelná výhoda.

Další věc, kterou by bylo vhodné řešit, je bezpečnost. Pokud budeme opět uvažovat skupinu, která čítá 100 členů, tak stačí zachytit alespoň jednu zprávu z 99. Vzhledem k tomu, že komunikace není žádným způsobem šifrovaná, tak by měl případný útočník velmi snadnou cestu k posílaným informacím a datům. Díky tomu, že komunikace mezi jednotlivými členy probíhá jako unicast, tak by šlo spojení teoreticky tunelovat přes SSH (pokud by o sobě všichni členové skupiny od začátku věděli). Jedná se ale o nouzové a nevhodné řešení. Mnohem vhodnější by bylo data šifrovat přímo v aplikaci.

Netroufám si tvrdit, že některá z dvou výše zmiňovaných variant je lepší. Provozovat tento navržený atomický multicast je z technologického hlediska možné všude tam, kde je možné použít IP multicast. Tvrdit to stejné obráceně není možné. Každopádně pokud má PlanetLab sloužit k testování a vývoji nových síťových technologií, tak bude nepochybně třeba nějakým způsobem vyřešit možnost používání IP multicastu na uzlech PlanetLabu.

6.8 Použití v praxi

Vzhledem k tomu, že výše uvedený návrh a implementace jsou realizovány na aplikační úrovni, tak předpokládám, že v každé aplikaci bude tento návrh implementován dle požadavků dané aplikace. Výše realizovaná implementace je jen jednou z možných variant. Pro názornost a jednoduchost je implementace realizována právě v jazyku C a s využitím BSD socketů. Další výhodou implementace v jazyku C bylo snadné spuštění na uzlech PlanetLabu (nebylo třeba nic doinstalovat). Formát zpráv, jejich velikost a další parametry závisí na požadavcích konkrétní aplikace. Není problém implementovat výše zmíněný návrh např. v jazyku Java nebo Perl.

Právě o implementaci v Perlu jsem zprvu také hodně uvažoval, protože Perl je standardně přítomen na všech uzlech PlanetLabu. Ke komunikaci by bylo rovněž využito TCP/IP a sockety (jejich varianta v Perlu). Níže je demonstrativně uvedena implementace socketu v Perlu.

```
use IO::Socket;
my $sock = new IO::Socket::INET (
LocalHost => 'nodeX', LocalPort => '15358', Proto => 'tcp', Listen => 1,
Reuse => 1, );
die "Could not create socket: $!\n" unless $sock;
```

Nakonec jsem se rozhodl pro jazyk C z důvodu již zmíněné názornosti.

Z výše uvedeného vyplývá, že implementovat navržené řešení je možné v širokém spektru jazyků a na různých platformách. Např. pod operačním systémem Windows by bylo možné tento návrh implementovat v jazyku C a ke komunikaci použít TCP/IP a WinSockety.

Závěr

Cílem této bakalářské práce bylo seznámit se s prostředím PlanetLab, způsobem vytváření aplikací v tomto prostředí a dále bylo cílem navrhnout a implementovat vhodnou aplikaci typu klient/server, která pracuje v prostředí PlanetLab.

Vzhledem k nutnosti nejdříve nastudovat dokumentaci a analyzovat všechny možnosti, výhody a nevýhody PlanetLabu, tak z tohoto důvodu nebylo možné zvolit vhodnou aplikaci již při zadání této práce. Postupnou analýzou prostředí PlanetLabu byla vhodná aplikace vybrána.

V teoretické části práce jsem nejprve rozebral problémy dnešního Internetu a jejich dopady na funkčnost samotného Internetu v současnosti, ale také v budoucnu. Tyto problémy daly vzniknout různým testovacím a vývojovým prostředím, které tyto problémy analyzují a snaží se nabídnout řešení. Jedná se zejména o testbeds a overlays. Následně jsem srovnal známé testbeds a overlays a popsal jejich možnosti. Jedním z těchto prostředí je právě PlanetLab, kterým jsem se dále zabýval do hloubky.

Následuje specifikace PlanetLabu, popis nejdůležitějších pojmů a celkového pojetí architektury. Detailně jsem se snažil popsat způsob práce, připojení a využívání zdrojů PlanetLabu. Z této části by mělo být pro čtenáře jasně pochopitelné, co PlanetLab vlastně je, jaké možnosti teoreticky nabízí a jak je s ním následně možné pracovat.

Bylo také nutné zanalyzovat PlanetLab po stránce použitých technologií, protože tyto použité technologie mohou pro koncové uživatele PlanetLabu skrývat specifická omezení, se kterými musí počítat při práci. Jednotlivé použité technologie jsem tedy zdokumentoval, aby bylo možné pro PlanetLab navrhovat a implementovat aplikace.

Myslím, že se mi podařilo prostředí PlanetLab popsat a specifikovat tak, aby bylo možné s ním pracovat po přečtení této práce.

V praktické části práce jsem se zabýval návrhem a implementací vhodné aplikace pro PlanetLab. Jedním z omezení PlanetLabu je nemožnost využít IP multicast. Rozhodl jsem se tedy navrhnout a implementovat variantu atomického multicastu, kterou by bylo možné používat v prostředí PlanetLabu. Obě varianty multicatu jsem srovnal včetně kladů a záporů.

Hlavní přínos této práce vidím ve zjištění a zdokumentování možností PlanetLabu, které se touto prací snažím předat všem potencionálním uživatelům. V budoucnu by bylo vhodné PlanetLab využít k vývoji a nasazení distribuované aplikace, která by plně využila potenciál PlanetLabu.

Při vytváření této práce jsem musel řešit mnohé problémy a seznámit se s řadou pro mě nových technologií a poznatků, což hodnotím jako velký přínos. Mnohé z těchto poznatků využiji při dalším studiu a také ve svém současném, ale i budoucím zaměstnání.

Literatura

[1] Navrátil, J. *PlanetLab - model budoucího Internetu.* Zpravodaj ÚVT MU. ISSN 1212-0901, 2006, roč. XVI, č. 5, s. 1-5.

[2] Chang, J. M., Maxemchuk, N. F. *Reliable Broadcast Protocols* , Murray Hill, AT&T Bell Laboratories 1984.

[3] Coulouris, G., Dollimore, J., Kindberg, T. *Distributed Systems: Concepts and Design*. 4. vydáni, New York, Pearson Education 2005. ISBN 0321263545.

[4] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*, New York, CRC Press 2006. ISBN 1584885645.

[5] Stevens, W. R., Fenner, B., Rudoff, A. M. *UNIX Network Programming*. 3. vydání, Boston, Addison-Wesley 2004. ISBN 0131411551.

[6] Anderson, T., Peterson, L., Shenker, S., Turner J. *Overcoming the Internet Impasse through Virtualization*, Washington, IEEE Computer Society 2005.

[7] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, Mike Wawrzoniak. *Operating System Support for Planetary-Scale Network Services* [online]
URL: http://www.planet-lab.org/

[8] *Projekt Internet2 – advanced networking consortium* [online]
URL: http://www.internet2.edu/

[9] *Emulab – network testbed* [online]
URL: http://www.emulab.net/

[10] *X-Bone – overlay testbed* [online]
URL: http://www.isi.edu/xbone/

[11] *DETER – cyber-security testbed* [online]
URL: http://www.isi.edu/deter/

[12] Matyska, L. *Techniky virtualizace počítačů* [online]
URL: http://www.ics.muni.cz/zpravodaj/articles/545.html

[13] *PlanetLab Hello World tutorial* [online]
URL: http://www.cs.princeton.edu/~jbagdis/planetlab/hello_world.pdf

[14] *Myrinet – high speed networking technology* [online]
URL: http://www.myricom.com/myrinet

[15] *The Coral Content Distributed Network project* [online]
URL: http://www.coralcdn.org/

[16] *The OceanStore project* [online]
URL: http://oceanstore.cs.berkeley.edu/

[17] *CoDeeN - academic testbed Content Distribution Network* [online]
URL: http://codeen.cs.princeton.edu/

[18] *DHARMA: Distributed Home Agent for Robust Mobile Access* [online]
URL: http://dharma.cis.upenn.edu/

[19] *Server farm* [online]
URL: http://www.webopedia.com/TERM/s/server_farm.htm

[20] *Linux-VServer: virtual machines in Linux* [online]
URL: http://www.linux-vserver.org/

[21] *Virtualizace v Linuxu* [online]
URL: http://www.root.cz/clanky/klonovanie-tucniaka-alebo-virtualizacia-v-linuxe-uvod/

Seznam příloh

Příloha 1. Průvodce vytvořením a spuštěním jednoduché aplikace v PlanetLabu.

Příloha 2. Nejznámější projekty využívající PlanetLab.

Příloha 3. Manuál k aplikaci.

Příloha 4. CD obsahující zdrojové kódy.

Obsah CD

CD má následující adresářovou strukturu:

- ./source - zdrojový kód a makefile k aplikaci
- ./deployment - skripty a soubory k testování a nasazení aplikace
- ./doc - technická zpráva ve formátu pdf
- ./hello_world - aplikace vypisující Hello World

Příloha 1. Průvodce vytvořením a spuštěním jednoduché aplikace v PlanetLabu

Tento jednoduchý návod má sloužit novým uživatelům PlanetLabu. Jedná se o jednoduchého průvodce, který provede uživatele krok za krokem od vytváření uživatelského účtu až po spuštění samotné aplikace. Návod v žádném případě nenahrazuje dokumentaci k PlanetLabu, která je dostupná na webových stránkách http://www.planet-lab.org v sekci Doc. Účelem tohoto průvodce je demonstrovat jednoduchost celého procesu práce s PlanetLabem.

Samotná aplikace je naprosto triviální. Jedná se o program napsaný v C a tento program vypíše Hello World.

Požadavky

Jediné, co budete potřebovat, je operační systém s SSH, respektive SCP klientem. Jednoduše řečeno, postačí téměř jakýkoli UNIX, případně i Windows (možnost použít Putty).

Vytvoření účtu

První věc, kterou je třeba udělat, je vytvořit si v PlanetLabu účet. Je to velmi jednoduché a účet si vytvoříte na stránkách PlanetLabu http://www.planet-lab.org. Před vytvořením účtu je nejprve třeba si přečíst podmínky užívání AUP (viz. podkapitola 2.2, případně sekce AUP přímo na webových stránkách). Po přečtení podmínek stačí kliknout na “Create an account”. Po vyplnění požadovaných informací klikněte na tlačítko “Register”. Váš požadavek je následně odeslán příslušnému PI (Principal Investigator, viz. podkapitola 2.1). Až PI vytvoří váš účet, tak vám zašle potvrzující email s uživatelským jménem a heslem.

Vytvoření SSH klíče

K připojení k libovolnému uzlu PlanetLabu budete u SSH potřebovat ověření pomocí veřejného klíče. Pokud zatím nemáte pár SSH klíčů, případně chcete použít k PlanetLabu jiné klíče, než které používáte dosud, tak tyto klíče vytvoříte následujícími příkazy (tento průvodce dále předpokládá, že privátní klíč bude vytvořen v `~//.ssh/id_rsa`):

```
$ ssh-keygen -t rsa -f  ~//.ssh/id_rsa
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): <enter some passphrase>
Enter same passphrase again: <enter your passphrase again?>
```

Nyní je třeba nahrát veřejný klíč do databáze PlanetLabu. Je to velmi jednoduché. Stačí se na stránkách PlanetLabu přihlásit pomocí vašich přihlašovacích údajů, které již máte. Po té stačí kliknout na "My Account". V sekci "Keys" vyberte "Manage Keys", dále klikněte na procházet a zadejte cestu k veřejnému klíči (pravděpodobně `~//.ssh/id_rsa.pub`). Potvrďte kliknutím na "Upload". Klíč by se měl objevit v seznamu všech vašich klíčů.

Vytvoření slice

Veškeré zdroje (veškeré nám dostupné systémové prostředky PlanetLabu) jsou přístupné pomocí abstrakce "slice". Slice je skupina všech zdrojů na uzlech PlanetLabu. Když přiřadíme uzel ke slice, tak na daném uzlu dojde k vytvoření virtuálního stroje. Pokud uzel ze slice odebereme, tak dojde ke zrušení virtuálního stroje na daném uzlu.

Váš PI (Principal Investigator, viz. podkapitola 2.1) je zodpovědný za vytváření slice. PI vás přiřadí k již existujícímu slice, případně vám vytvoří nový. Oboje je záležitost několika minut. Pokud máte slice již přiřazený, respektive vytvořený a přiřazený, tak po zalogování pomocí vašich přihlašovacích údajů vyberete v menu "Slices". Tam uvidíte seznam všech dostupných slices, které jsou vám k dispozici.

Přiřazení uzlů ke slice

Abychom mohli s uzly pracovat, musíme je nejdříve ke slice přiřadit. Nejjednodušší je přiřadit uzly po jednom přímo na webových stránkách PlanetLabu v sekci "Slices" a dále "Manage nodes". Existuje další varianta a tou je použití CoMon (viz. podkapitola 6.3 CoDeeN) a PlanetLab API (viz. podkapitola 3.5).

Pro tento jednoduchý návod plně stačí první varianta.

Nasazení aplikace na uzly

Je několik způsobů jak nasadit aplikaci na uzel, respektive uzly. Pro nasazení aplikace na jeden uzel stačí použít SCP. Pokud budeme chtít aplikaci nasadit současně na více uzlů, tak můžeme použít parallel SSH (http://www.theether.org/pssh/). Nevýhodou tohoto řešení je, že všechny příkazy jsou

spouštěny paralelně a tak může v jednom okamžiku dojít např. k velkému vytížení zdroje (např. přetížení web serveru). Nejlepším řešením pro nasazení aplikace na více uzlů je použití CoDeploy (viz. Příloha 2, CoDeeN).

Pro tento jednoduchý návod plně dostačuje první varianta klasického shellu a použití SCP:

```
$ scp -i ~/.ssh/id_rsa -r hello_world
cesnet_vutbr2@planetlab1.cesnet.cz:
```

Nyní máme naši jednoduchou aplikaci na uzlu planetlab1.cesnet.cz. Teď se stačí přihlásit a aplikaci spustit:

```
$ ssh -l cesnet_vutbr2 -i ~/.ssh/id_rsa planetlab1.cesnet.cz
$ ./hello_world
Hello World
```

Příloha 2. Nejznámější projekty využívající PlanetLab

Na první pohled se může zdát, že PlanetLab zatím není v praxi nijak uplatněn, případně že žádné velké korporace PlanetLabu nevyužívají. Opak je ovšem pravdou. Mezi členy PlanetLabu patří např.:

- Intel, privilegované členství
- Hewlett Packard, privilegované členství
- Google, plné členství
- AT&T, asociativní členství
- France Telecom, asociativní členství
- AT Corporation, asociativní členství
- DoCoMo Communications Laboratories USA, asociativní členství
- Lucent - Bell Labs, asociativní členství
- NEC Laboratories, asociativní členství
- Telecom Italia, asociativní členství

Google a Intel v PlanetLabu testovaly systém několika set tisíc kamer. Dále Google v rámci PlanetLabu realizuje otevřený projekt, který vyhodnocuje dostupnost jeho serverů z různých částí světa.

Za nejzajímavější projekty s velkou perspektivou se dají označit:

- OceanStore, URL: http://oceanstore.cs.berkeley.edu/
- Coral, URL: http://www.coralcdn.org
- CoDeeN, URL: http://codeen.cs.princeton.edu/
- RON (Resilient Overlay Network), URL: http://nms.csail.mit.edu/ron/
- DHARMA (Distributed Home Agent for Remote Mobile Access), URL: http://dharma.cis.upenn.edu/

OceanStore

OceanStore je globální perzistentní úložiště dat pro miliardy uživatelů, které je postaveno na infrastruktuře nedůvěryhodných serverů, ale přesto poskytuje konzistentní, vysoce dostupnou a spolehlivou možnost uložení dat.

Členem OceanStore se může stát v podstatě každý. Stačí poskytnout úložiště (např. klasické PC s několika pevnými disky). Pro běžného uživatele se stačí zaregistrovat u libovolného poskytovatele služeb OceanStore, ale tento užvatel může využívat kapacitu úložiště i konektivitu ostatních poskytovatelů služeb OceanStore. Jednotliví poskytovatelé si mezi sebou prodávají kapacitu úložiště

a konektivitu, transparentně směrem k uživatelům. Díky tomuto principu fungování celého systému OceanStore poskytuje větší kvalitu poskytovaných služeb (QoS), než jakákoli samostatná společnost.

Data jsou cachována náhodně a tak každý server může kdykoli vytvořit lokální kopii datového objektu. Tyto kopie poskytují rychlejší přístup i větší robustnost, navíc snižují zahlcení díky možnosti lokalizovat síťový provoz přístupu k daným objektům.

Pro zabezpečení dat na jednotlivých serverech (ať se jedná o pád systému, selhání hardware, případně únik dat a neoprávněný přístup) jsou použity kryptografické algoritmy a algoritmy řešící redundanci dat. O vysokou konzistenci jednotlivých kopií datových objektů se stará Byzantine-fault tolerant commit protokol. Každá aplikace, která potřebuje zvýšit výkon a dostupnost dat, může snížit konzistenci pomocí OceanStore API.

OceanStore ukládá každou verzi datového objektu v neměnné podobě (objekt je read-only). Každý takový objekt je enkódován pomocí tzv. erasure code a následně je rozšířen na stovky nebo tisíce serverů. Několik takto enkódovaných fragmentů stačí k rekonstrukci celého datového objektu. Z toho plyne, že jen opravdu globální živelná pohroma je schopna zničit data uložená v systému OceanStore (bylo by zapotřebí vyřadit většinu serverů, což je vzhledem k jejich počtu a rozmístění po celém světě téměř nereálné).

Introspektivní vrstva přizpůsobuje celkové chování systému s cílem zlepšit výkonnost a toleranci selhání. Interní mechanismy systému OceanStore monitorují síťový provoz a dostupnost volných prostředků.

Většina komponentů, ze kterých se OceanStore skládá, již funguje izolovaně. Celkový prototyp je zatím stále ve vývoji. OceanStore je testován a vyvíjen v PlanetLabu.

OceanStore je implementován v prostředí Java a je ke stažení na webových stránkách http://oceanstore.sourceforge.net/.

Coral

CoralCDN je peer-to-peer distribuční síť, která se skládá z proxy serverů a jmenných serverů rozmístěných po celém světě. CoralCDN umožňuje provozovat web server, na který jsou kladeny vysoké nároky z hlediska konektivity a dostupnosti např. na obyčejné adsl lince.

Použití CoralCDN k tomuto účelu je velmi jednoduché. Stačí k vybrané adrese (např. www.vutbr.cz) přiřadit řetězec .nyud.net (výsledek tedy bude www.vutbr.cz.nyud.net). Peer-to-peer DNS servery přesměrují připojení na některý z proxy serverů, který ve své cache má požadované stránky a tím minimalizuje nároky na konkrétní web server. Ke komunikaci jednotlivých proxy serverů jsou použity efektivní peer-to-peer technologie, které zajišťují rychlé vyhledání požadované stránky, pokud existuje na libovolném proxy serveru kdekoli v síti.

CoralCDN se snaží zajistit, aby v jeho infrastruktuře nedocházelo k přetěžování některých jeho serverů. Používa k tomu distributed sloppy hash table (DSHT), která vytváří clustery serverů, které mezi sebou přenášejí požadované informace a rovnoměrně rozdělují zátěž.

Jednotlivé proxy servery a DNS servery jsou právě uzly PlanetLabu, na kterých celý projekt CoralCDN běží.

CoDeeN

CoDeen je akademický testbed (konkrétně jde o Content Distribution Network, tzn. síť distribuující obsah), který je postaven na PlanetLabu. Projekt vyvíjí Network System Group (http://www.cs.princeton.edu/nsg/) na Princeton University. Tento testbed využívá uzlů PlanetLabu jako výkonných proxy serverů, které jsou rozmístěny téměř po celém světě. Tyto proxy servery mezi sebou neustále komunikují a díky tomu nabízí uživatelům rychlé a robustní služby.

CoDeeN zaštiťuje ještě další zajímavé projekty:

- CoBlitz: jde o službu sloužící k distribuci velkých souborů prostřednictvím HTTP, tyto soubory jsou distribuovány právě přes uzly PlanetLabu.
- CoDeploy: jde o synchronizační nástroj, který slouží k distribuci souborů / aplikací na jednotlivé slice PlanetLabu.
- CoDNS: jedná se o velmi rychlé a spolehlivé DNS servery; každý uzel PlanetLabu funguje jako DNS server.
- CoMon: jde o webovou službu, díky které je možné sledovat stav jednotlivých uzlů PlanetLabu.
- CoTest: jedná se o testovací a debugovací nástroj, který řeší problém s přihlašováním na jednotlivé uzly PlanetLabu.
- CoVisualize: jde o vizualizační nástroj, který graficky znázorňuje aktivitu uzlů PlanetLabu.

RON (Resilient Overlay Network)

Projekt RON byl založen s cílem zlepšit robustnost a dostupnost Internetových linek mezi jednotlivými stroji v Internetu. Hlavní myšlenkou je vyvinout postupy, které by umožnily jednotlivým koncovým strojům v Internetu (a aplikacím běžícím na těchto strojích) zvýšit spolehlivost a výkon samotných datových přenosů, které jsou realizovány právě přes Internet. Jednotlivé uzly projektu RON analyzují stav a propustnost linek mezi sebou a dále mezi ostatními uzly Internetu. Podle výsledků analýzy pakety putují přímo na další uzly Internetu nebo jsou směrovány přes uzly projektu RON.

Uzly projektu RON jsou vlastně uzly PlanetLabu a díky tomu je pokrytý opravdu téměř celý svět. Z toho plyne, že systém má přehled o propustnosti a stavu velké většiny linek Internetu, což běžné Internetové routery rozhodně nemají.

RON je overlay vůči současnému systému routování v Internetu.

DHARMA (Distributed Home Agent for Robust Mobile Access)

Většina mobilních uzlů využívá k poskytování svých služeb overlay na IP vrstvě a s tím spojený systém doručování IP paketů. Bezdrátové spoje (připojení mobilního zařízení do routované IP sítě) ovšem trpí nedostatky jako nedostupnost sítě, špatná kvalita signálu, výpadky signálu, rušení atd.

Právě tyto nedostatky jsou velkým problémem pro TCP/IP protokol, protože při velké ztrátovosti paketů drasticky klesá i výkonnost. Další problém, který se může vyskytnout, je neefektivní triangulární routování (pokud je mobilní host příliš daleko od "domova", tak routování s mobilní IP může být neefektivní).

DHARMA řeší oba tyto problémy. Problémy nespolehlivé bezdrátové linky řeší několikanásobné linky a algoritmy, které vybírají dostupné linky podle požadavků aplikací a podle možností sítě. Problém mobilní IP a velké vzdálenosti od "domova" řeší vytvářením mnohonásobných "domovů", které jsou vytvářeny v overlay sítích, jako je právě PlanetLab.

DHARMA je tedy určena všem, kteří často cestují a připojují své mobilní zařízení na různých místech, mají nestabilní linky, případně často zapínají/vypínají svoje mobilní zařízení.

Transportní vrstva u projektu DHARMA je implementována tak, že podporuje kompletní škálu TCP aplikací. TCP připojení může zůstat navázáno i pokud ztratíme síťové připojení (např. špatný signál), změníme zdroj konektivity (např. z Wifi na GPRS), případně suspendujeme svoje mobilní zařízení.

Projekt je ke stažení na webových stránkách http://dharma.cis.upenn.edu/dl/.

Příloha 3. Manuál k aplikaci

Požadavky

Program běží na jakémkoli operačním systému Linux. Program ke svému přeložení potřebuje knihovny `sys/types.h`, `sys/socket.h`, `netinet/in.h`, `arpa/inet.h`, `ctype.h`, `err.h`, `netdb.h`, `signal.h`, `stdio.h`, `stdlib.h`, `string.h`, `unistd.h`, `sys/time.h`, `time.h`, `sys/shm.h`, `sys/stat.h`, `semaphore.h` a `pthread.h`.

Instalace

Program není potřeba instalovat. Stačí zkompilovat zdrojový soubor a program spustit.

Ovládání

Aplikace je čistě konzolová a její ovládání je velmi snadné. Program je možné spustit s třemi volitelnými parametry. Parametry a jejich syntaxe jsou (-- help vypíše nápovědu):

```
amul [-p <port>] [-r <reachability>] [-f <soubor>]
```

- Port (volitelný) určuje číslo portu, na kterém bude člen přijímat zprávy; při nezadání portu je implicitně nataven port 15358.
- Reachability (volitelný) určuje počet pokusů o doručení zprávy dalším členům; při nezadání hodnoty je implicitně nastavena hodnota 5.
- Soubor (volitelný) určuje soubor, do kterého se ukládají přijaté zprávy; při neuvedení se zprávy neukládají.

Po spuštění aplikace načítá z klávesnice zprávu. Zadávání zprávy je ukončeno znakem ".". Následně po stisknutí klávesy "Enter" je zpráva odeslána všem známým členům a je také zobrazena na stdout. Příchozí zprávy od ostatních členů jsou taktéž zobrazovány na stdout. Aplikace se ukončuje zadáním zprávy ve formátu "exit.".

Aplikace při spuštění zkontroluje, zda existuje soubor `ip_list`, ve kterém mohou být uloženy informace o ostatních členech (konkrétně IP adresa a port). Pokud tento soubor existuje, aplikace načte tyto informace o ostatních členech. Na každém řádku je v souboru uložena IP adresa jednoho člena, která je zakončená znakem "#". Může být volitelně uveden i port (oddělený znakem ":" od IP adresy; za číslem portu následuje znak "#"). Jinak je implicitně k tomuto členovi nastaven port 15358.