Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Distribuce kódu v aktivních sítích

Plzeň, 2009 Bc. Petr Štěpánek

… originál zadání …

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 20. května 2009 ……..……

Petr Štěpánek

Abstract

Active networks define a networking architecture, where a code execution is an integral part of entire network design. In contrast to traditional networking, this brings a whole new level of abstraction and possibilities. An active network is no longer only a simple connection between static nodes. In active networks applications are allowed to move from node to node the same way data do. Developers can use this to simply deploy new protocols and capabilities in the network. The aim of this diploma thesis is to analyze ways a code distribution protocol can be implemented in an active network and describe the process of implementation of such protocol in Smart Active Node (SAN) project.

Keywords: active networks, code distribution, SAN

1. Úvod

Vývoj počítačových sítí, započatý na konci šedesátých let v USA státní agenturou DARPA, ovlivnil postupně mnoho oblastí lidské činnosti. Tento vývoj nelze v žádném případě považovat za ukončený. Jednou z možných cest k efektivnějším počítačovým sítím je právě myšlenka aktivní sítě.

Cílem této práce je popsat jednu z důležitých součástí každé aktivní sítě, protokol pro distribuci kódu, jeho implementaci v projektu Smart Active Node (SAN) a ukázat výsledky, kterých bylo dosaženo.

2. Aktivní síť

Tato kapitola si klade za cíl popsat základy konceptu aktivních sítí, ukázat hlavní rozdíly mezi klasickým přístupem a aktivními sítěmi a zhodnotit možnosti využití aktivních sítí spolu s jejich výhodami a nevýhodami.

2.1. Principy

Aktivní síť slouží ke spojení mezi uzly v síti a výměně dat mezi nimi podobně jako jakákoli jiná počítačová síť. Základní změnou oproti klasickým počítačovým sítím je to, díky čemu jsou tyto sítě označovány jako aktivní. Aktivitou je myšleno vykonávání kódu na jednotlivých uzlech na základě přijatých dat, možnost modifikovat tato data, případně i chování uzlu, a také možnost přenosu nejen dat, ale i samotného kódu.

To je umožněno díky prostředí vytvořeném na aktivním uzlu, které umožňuje vykonávání distribuovaného kódu, a dále také způsobem přenosu dat. Jednotkou přenosu je tzv. kapsle. Kapsli si lze představit jako balík dat a s nimi souvisejících informací. Obdobně jako paket v IP síti i kapsle má záhlaví s podobným obsahem a významem. Zároveň je s každou kapslí svázán kód, ke kterému kapsle náleží. To je obdobou id protokolu, popř. čísla portu TCP/UDP, které také určuje aplikaci. Vazbu mezi kapslí a kódem lze realizovat dvěma způsoby:

- Kód je přímo součástí kapsle – Kapsle pak tvoří samostatnou jednotku a může být nezávislá na okolí. Akce definované programovým kódem svázaným s kapslí, jsou přenášeny jako její nedílná součást. Výhodou tohoto přístupu je možnost s kapslí pracovat ihned po přijetí na uzel a odpadají komplikace s hledáním požadovaného kódu. Nevýhodou je pak velikost přenášených dat, kdy kód přenášený kapslí nesmí být neúměrně veliký a zatěžovat tak přenosové kapacity.
- Kód je distribuován jiným kanálem – Kapsle obsahuje pouze identifikátor příslušné aplikace, a pokud mají být na uzlu zpracována data kapsle, je nutné aplikaci na uzel nainstalovat nebo získat jinou cestou. Ideální řešení je

automatická a transparentní distribuce z jiného uzlu sítě, kde je již aplikace nainstalována.

Díky použití kapslí se tak aktivní sítí kromě dat přenáší obdobným způsobem i kód a tím je možné flexibilně měnit funkcionalitu jednotlivých uzlů podle aktuálních požadavků uživatelů.

V aktivní síti existují dva druhy kódu:

- Aktivní aplikace – Aplikace nainstalované na uzel, ať již samotným uživatelem nebo automaticky. Aplikace spouští buď uživatel, nebo se spustí v rámci vykonávání jiné aplikace či kódu kapsle. Doba běhu se předpokládá delší než u kódu kapsle (viz níže).
- Kód kapsle – Většinou menší rozsahem, než aktivní aplikace (s ohledem na častější přenos v aktivní síti). Kód je vykonáván při zpracování dat kapsle, která přišla na uzel.

Oba druhy kódu lze přenášet mezi uzly a slouží obdobně jako aplikace v klasické síti k výpočtům a operacím nejrůznějšího druhu. Způsob využití aktivních sítí je popsán v následující kapitole 2.2.

S výše popsaným způsobem přenosu a vykonávání kódu v aktivní síti vznikají kromě nových možností také nové hrozby. Proto je, oproti klasickým sítím, již od počátků návrhu velmi důležitou součástí bezpečnost. Mezi největší nebezpečí patří především:

- Podvržení škodlivého kódu – Uzel poskytuje prostředí pro běh kódu, který přichází z ostatních uzlů a o kterém uzel nemá předem žádné informace. V aktivní síti tedy musí existovat možnost, jak ověřit původ kódu, omezit přístup z kódu ke zdrojům uzlu a ostatnímu kódu, případně může uzel dokonce sledovat vykonávání kódu a zamezit vykonávání podezřelých instrukcí.
- Modifikace kapslí – Kapsle musí poskytovat mechanismus, jak ověřit, že jejich obsah nebyl změněn a že udávaný zdroj kapsle a příslušející aplikace odpovídá originálu.

- Kompromitace uzlu aktivní sítě – Musí existovat mechanismus identifikace jednotlivých uzlů v síti a zároveň také chráněn přístup k uzlům a na nich nainstalovaným aplikacím.

2.2. Možnosti využití aktivních sítí

Aktivní sítě vznikly jako odpověď na problémy a nedostatky klasických počítačových sítí.

Jedním z nich je nedostatečná flexibilita při zavádění nových služeb a protokolů. Aktivní sítě jsou od začátku zamýšleny jako prostředí pro uskutečňování rychlé a efektivní změny služeb přítomných na jednotlivých uzlech sítě. Tím odstraňují nákladné a časově náročné spravování jednotlivých uzlů, o které se v aktivních sítích starají samotné aplikace. Uzly aktivní sítě jsou schopné si sami zajistit požadovanou funkčnost, kterou stačí nejdříve nasadit na jeden uzel a dál se již podle potřeby rozšiřuje sama. Kromě nasazování zcela nové funkčnosti do sítě je samozřejmě stejně jednoduché i upravování stávajících aplikací.

Další výhodou aktivních sítí oproti sítím klasickým je možnost přenesení zátěže z jednoho uzlu na jiný. Spolu s přenosem kódu mezi uzly se samozřejmě může přenášet i kontext dané aplikace a na cílovém uzlu může výpočet pokračovat od místa přerušení běhu aplikace na zdrojovém uzlu, nebo může výpočet probíhat paralelně.

Decentralizovaná povaha aktivních sítí je také ideální pro často se měnící topologie sítě, např. sítě s velkým počtem mobilních stanic (uzlů). Podpora mobility je v aktivních sítích přítomna již v od prvního návrhu.

Z výše uvedeného vyplývá, že aktivní sítě poskytují velmi flexibilní infrastrukturu, která se snaží uživatele neomezovat a naopak jim umožňuje jednoduše využít prostředky dostupné kdekoliv v síti. Je jen na uživatelích, jak dokážou tyto možnosti využít.

2.3. Historie

Přesto, že myšlenka aktivních sítí není stará, můžeme již hovořit o historii aktivních sítí a jejich vývoji. Jako počátek aktivních sítí jsou uváděny roky 1994 a 1995 (viz [TEN97]), kdy agentura DARPA započala diskuzi s odborníky o problémech současných počítačových sítí. Od té doby vzniklo několik projektů zaměřených na různé oblasti využití aktivních sítí.

2.3.1. Projekt ANTS

Původní implementace konceptu aktivních sítí. Využívá jazyk Java. Hlavní nevýhodou je skoro neexistující zabezpečení a také výkon výsledného uzlu aktivní sítě. Hlavního cíle projektu, prokázat použitelnost aktivních sítí, bylo dosaženo (viz [WGT98] a [WGT99]).

2.3.2. Grade32

Grade32 je projektem Katedry informatiky a výpočetní techniky na Fakultě aplikovaných věd Západočeské Univerzity v Plzni. Vznikl s úmyslem vytvořit prostředí pro distribuované výpočty, ve kterém by bylo možné dynamicky přerozdělovat zátěž mezi jednotlivými účastníky výpočtu. Koncepce projektu vychází z myšlenky aktivních sítí a *nezabýval se některými detaily, které je nutné pro produkční činnost aktivních sítí řešit* ([Rej08], str. 13).

2.3.3. PLANet

Základem tohoto projektu je jazyk PLAN, který byl navržen speciálně pro využití v aktivních sítích typu PLANet. *Kromě standardních klíčových slov jako definování datových typů nebo pro potřeby větvení a cyklů, obsahuje další klíčová slova přímo svázaná s použitím v prostředí aktivního uzlu PLANet* ([Rej08], str. 13). Slouží k popisu aplikací, ale také pro implementaci směrovacího algoritmu a byl vytvořen s důrazem na bezpečnost aktivní sítě.

2.3.4. Mobilní agenti

Historicky podobnou (paralelní) větví vývoje počítačových sítí jsou mobilní agenti. Tato technologie si klade velmi podobné cíle jako koncept aktivních sítí. Hlavním rozdílem je ovšem přístup k síti samotné. Zatímco mobilní agenti využívají již existující síťové technologie (např. IP protokol) a nad nimi staví další vrstvu poskytující uživatelům širší možnosti využití, aktivní sítě mění celou síťovou infrastrukturu. Cílem aktivních sítí je nasazení nového síťového protokolu na úrovni, kde je nasazen IP protokol.

2.3.5. Ostatní

Jako další projekty uvádí M. Rejda ([Rej08], str. 21) např. SwitchWare ([Alex98a] a [Alex98b]), CANEs ([MB99]) nebo NetScript ([YDS96]).

2.4. Projekt SAN

SAN je implementací uzlu aktivní sítě, která začala vznikat v roce 2006 na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Vychází ze zkušeností výše uvedených projektů a jejím cílem je vytvoření nejen testovací platformy pro aktivní sítě, ale i vyřešení otázek, které zůstaly otevřené po skončení finanční podpory ze strany DARPA. Potencionálním cílem je tak i skutečné nasazení SANu na síťová zařízení. Pro implementaci prototypu byl zvolen programovací jazyk Java, v současné době i C++. Obsáhlá dokumentace první verze projektu je obsažena v diplomové práci Uzel aktivní sítě od M. Rejdy ([Rej08]).

3. Cíle a motivace

Tato kapitola se zabývá tématem diplomové práce z pohledu jejího cíle. Vysvětluje a zdůvodňuje cíle, které si práce klade a jejich motivaci.

3.1. Motivace

Jak je z předchozího popisu principu aktivních sítí patrné, implementace aktivní sítě se neobejde bez možnosti distribuovat mezi uzly sítě kód (dále je pro kód používáno také označení aplikace nebo balík kódu). Tato funkcionalita je pro aktivní sítě klíčová a je tím, co aktivní síť odlišuje od sítí tradičních (spolu s možností kód vykonávat). Je tedy bezpodmínečně nutné věnovat tomuto problému dostatečnou pozornost.

Distribuce kódu

Distribuci kódu je proces, který v aktivní síti zaručuje, že aplikace může být vykonána na libovolném uzlu i přesto, že se na daném uzlu v současné době kód aplikace nenachází. Lze si ji zjednodušeně představit jako přenos aplikace mezi zdrojovým uzlem a cílovým uzlem s cílem umožnit vykonání aplikace na cílovém uzlu. Distribuce kódu je tedy inicializována potřebou vykonat kód aplikace, která není přítomna na uzlu.

3.2. Cíle práce

Cílem práce je navrhnout a implementovat vylepšení protokolu pro distribuci kódu, který umožní provádět distribuci kódu automaticky a pro uživatele sítě zcela transparentně, bez ohledu na složitost aplikace sestavené z programových modulů.

Lze si představit mnoho různých způsobů, jak distribuci kódu provádět. Jejich analýze se věnuje kapitola 4. Hlavními cíly distribuce kódu jsou spolehlivost, efektivita a bezpečnost. Spolehlivostí se rozumí schopnost najít v síti požadovanou aplikaci, přenést ji na cílový uzel, úspěšně ji nainstalovat a spustit. Celá distribuce kódu by měla probíhat s co nejmenšími nároky na přenosové kapacity a ostatní zdroje uzlů v síti, tedy efektivně. A v neposlední řadě musí být při průběhu distribuce kódu zajištěna bezpečnost přenosu a znemožněno např. podvržení jiné aplikace.

4. Analýza

V této kapitole je popsána analýza problému, provedená před samotným začátkem implementace. Výsledkem analýzy je postup, jehož provedením (popsaném v následující kapitole 5) bylo dosaženo cíle samotné diplomové práce (viz předchozí kapitolu 3.2).

4.1. Úvodní přehled

Výchozím bodem analýzy byla definice distribuce kódu z kapitoly 3, vytvořená před začátkem projektu. Z té jasně vyplývá, že distribuce kódu není jednoznačně daný proces a lze ji navrhnout několika různými způsoby. Dále budou jednotlivé odlišnosti možných způsobů distribuce kódu rozebrány a bude vybrán ten nejvhodnější pro projekt SAN. Poslední část kapitoly o analýze již opouští teorii a týká se rozboru dopadů implementace distribuce kódu na samotné moduly uzlu aktivní sítě projektu SAN.

4.1.1. Účastníci distribuce kódu

Předně je to část uzlu implementující samotnou distribuci kódu. Její hlavní povinností je nalezení požadované aplikace a obsloužení požadavků na aplikace od ostatních uzlů.

Dále je to úložiště kódu, které aplikace poskytuje. V něm je nutné zajistit uložení aplikace, její zpřístupnění ostatním komponentám uzlu a v neposlední řadě možnost odstranění aplikace, protože velikost úložiště je omezená. Při odstraňování je nutné dbát na to, aby nedošlo k úplné ztrátě aplikace (tzn. aplikace musí být vždy na některém uzlu sítě přítomna).

4.2. Části procesu distribuce kódu

Distribuci kódu lze rozložit na čtyři základní části: inicializace, hledání zdrojového uzlu, přenos aplikace a její instalace na cílovém uzlu.

4.2.1. Inicializace

Jak již bylo uvedeno výše, distribuce kódu je inicializována nepřítomností požadovaného kódu aplikace na uzlu (tzn. na uzel dorazí kapsle, obsahující data pro aplikaci, která se na uzlu nenachází a zároveň je potřeba tato data aplikaci předat, nebo vykonat kód spojený s kapslí).

Ovšem můžeme uvažovat i o další variantě, kdy se aplikace může na uzel přenést před tím, než uzel zjistí, že ji potřebuje. Tím se ušetří nutnost odeslat z uzlu požadavek na aplikaci a její, potenciálně složité, vyhledávání. Jsou zde ovšem kladeny vyšší nároky na vlastnosti implementace distribuce kódu (pokud má distribuce kódu probíhat automaticky) a vzniká riziko omylů (mechanismus distribuce kódu špatně určí, zda se aplikace na uzlu již nachází a aplikace se bude přenášet zbytečně). A ani v této variantě nelze úplně vynechat schopnost uzlu nalézt požadovanou aplikaci, protože se nelze s jistotou spolehnout na dodání aplikace okolními uzly.

4.2.2. Nalezení vhodného uzlu s aplikací

Poté, co uzel zjistí, jakou aplikaci potřebuje, je nutné určit, který uzel o poskytnutí aplikace požádá. Z toho vyplývá, že v síti musí také existovat kapsle, která představuje žádost o aplikaci.

Tím se dostáváme k samotnému nalezení vhodného uzlu. Zde se nabízí více možností. Kvůli charakteru aktivních sítí je takřka vyloučeno použít centralizovanou metodu s jedním uzlem (nebo několika), který by se o distribuci aplikací staral. Přijatelnou možností by bylo např. udržovat tabulky aplikací přítomných na jednotlivých uzlech, periodicky je mezi uzly vyměňovat a tím udržovat ve všech uzlech informace o aplikacích přítomných na ostatních uzlech. Ovšem kvůli složitosti této metody se jako mnohem lepší pro použití v aktivních sítích jeví metoda

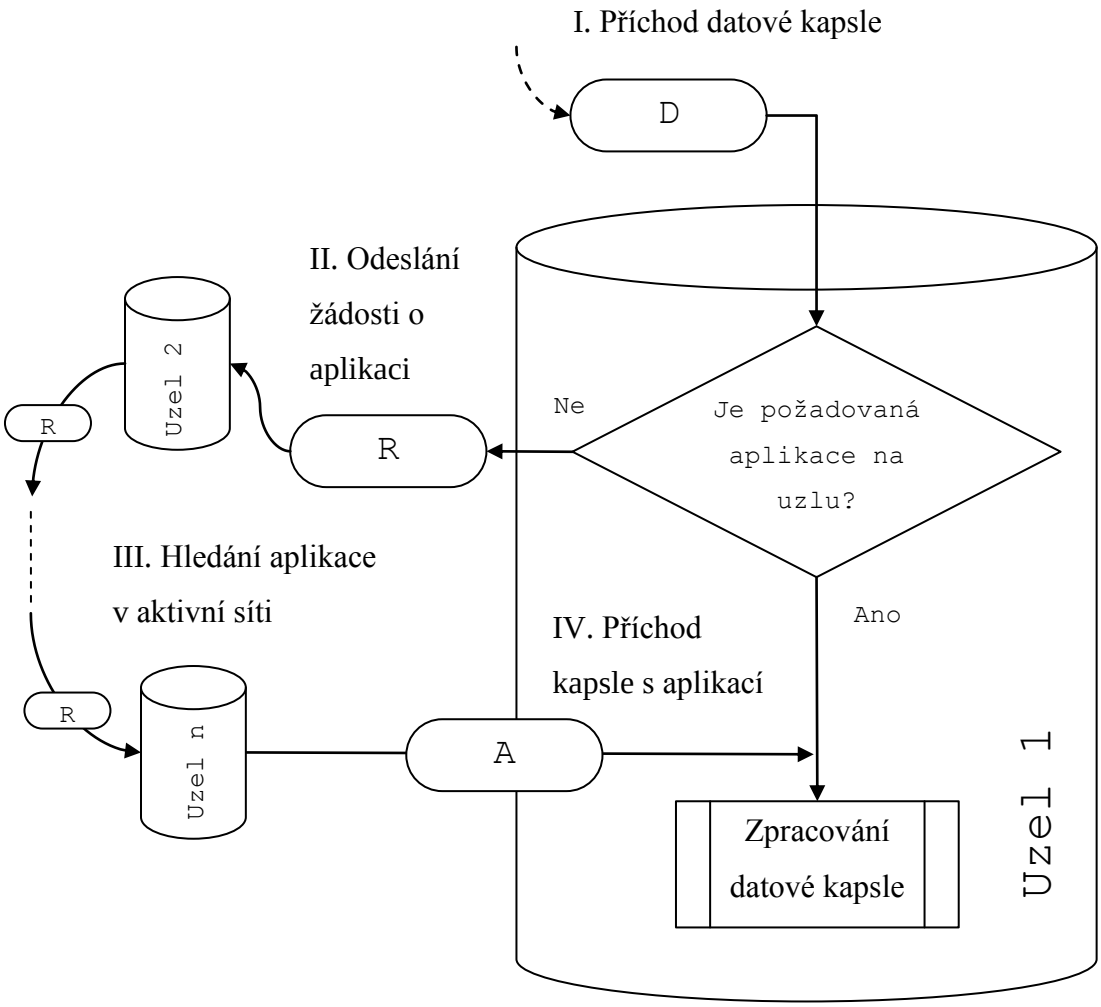
nezávislá na aktuálním spojení uzlů, kdy každý požadavek na aplikaci bude obsluhován samostatně a na základě aktuálních informací bude vybrán nejvhodnější uzel, který bude o aplikaci požádán. Protože nelze s naprostou jistotou vědět, že v daném okamžiku je na určitém, nejen sousedním, uzlu aplikace přítomna (předpokladem je, že každý uzel rozhoduje samostatně o tom, jaké aplikace bude uchovávat ve svém úložišti), nemusí být požadavek na aplikaci obsloužen cílovým uzlem, na který požadavek směřuje. Pak ovšem cílový uzel musí provést vše potřebné, aby žádající uzel aplikaci obdržel. To znamená především opětovné nalezení dalšího uzlu, na kterém se aplikace s velkou pravděpodobností nachází a předání požadavku na tento uzel. Tím vzniká řetěz požadavků, který by nakonec měl dosáhnout uzlu s požadovanou aplikací.

4.2.3. Přenos kódu

Samotný přenos aplikace je nejvhodnější provést kapslí se stejným formátem, jako má kapsle pro přenos dat, pouze místo dat bude jejím obsahem aplikace. Poté zbývá objasnit činnosti, které bezprostředně předchází a následují za přenosem. Přenos aplikace na uzel, který o ni požádal, v žádném případě nemusí znamenat její odstranění z úložiště. Naopak je žádoucí ponechat na uzlu kopii aplikace pro další případné použití. Po příchodu kapsle s kódem aplikace na uzel je nutné ho uložit a to může mít za následek odstranění kódu jiné aplikace z úložiště. Po přijetí kapsle s aplikací na cílovém uzlu je nutné aplikaci nainstalovat a tím umožnit její používání.

4.3. Shrnutí teorie

Výsledkem předchozích úvah je návrh protokolu distribuce kódu, který bude mít následující vlastnosti: Proces je inicializovaný až při potřebě získat aplikaci a distribuce aplikací nebude probíhat jindy než na vyžádání konkrétního uzlu. Uzel, ze kterého je aplikace distribuována, bude nalezen pomocí kapsle vyslané ze zdrojového uzlu a tato kapsle bude předávána mezi uzly až do nalezení aplikace. Proces distribuce kódu je znázorněn na obr. 2. Hlavní odpovědností procesu distribuce kódu je tedy rozhodnutí o tom, jaký uzel bude o aplikaci požádán.



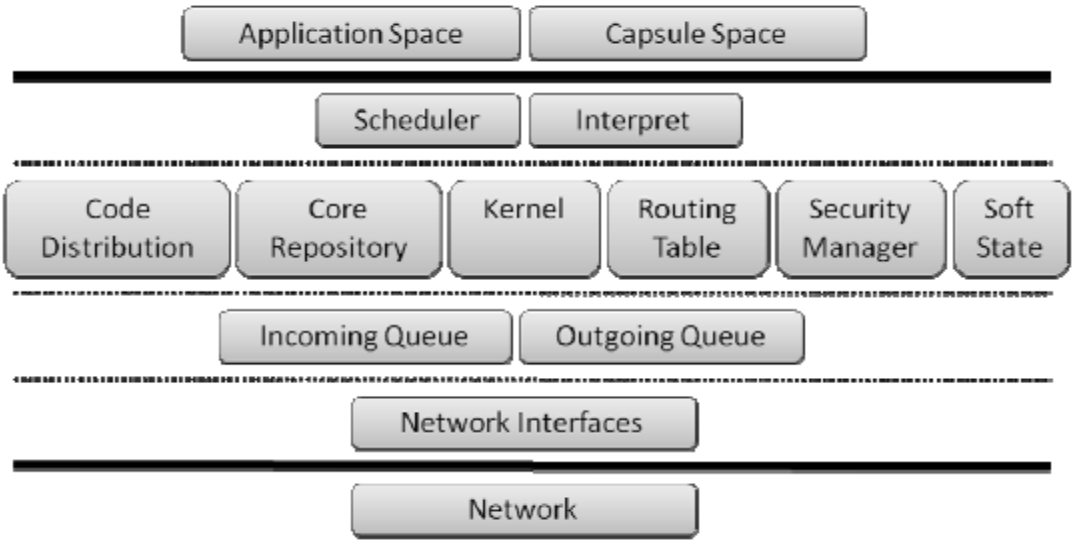
Obr. 1.: Schéma procesu distribuce kódu

4.4. SAN

Následuje popis analýzy možností implementace navrhovaného mechanismu distribuce kódu v uzlu sítě SAN. Distribuci kódu bylo nutné implementovat do již funkčního prototypu uzlu. Architektura uzlu je detailně popsána v [Rej08]. Zde jen stručné připomenutí.

4.4.1. Architektura uzlu SAN

Základní moduly uzlu SAN (schéma převzato z [Rej08]).



Obr. 2.: Architektura uzlu SAN

Celý uzel tvoří vrstvu mezi samotnou sítí a aplikacemi (spolu s kapslemi), běžícími na uzlu. Poskytuje aplikacím prostředí potřebné pro jejich běh a spojení s ostatními uzly aktivní sítě.

Poznámka: slovo modul je dále v popisu architektury uzlu SAN používáno ve smyslu množiny tříd a rozhraní, která dohromady definují určitou ohraničenou funkčnost uvnitř uzlu.

Následuje popis jednotlivých modulů z, jejich funkcí a zodpovědností.

- Scheduler – Zodpovídá za plánování běhu aplikací a kapslí na uzlu. Stará se o dodržování přidělených časových kvant a priorit jednotlivých běžících procesů. Od modulu Kernel dostává požadavky na vykonání kódu (aplikací nebo kapslí), samotný kód získává z modulů Code Repository a Code Distribution a kód je předáván k vykonání modulu Interpret, který je tímto modulem ovládán. Obsahuje fronty kapslí a aplikací čekajících na vykonání a případně i na kód, který není na uzlu nainstalován.

- Interpret – Vykonává kód aplikací a kapslí. Kód může být uložen ve formátu BSH nebo SPK balíčku, který v obou případech obsahuje zdrojový kód napsaný v jazyce Java verze 1.5.

- Code Distribution – Vlastnostmi tohoto modulu se zabývá celá tato práce. V původním stavu projektu SAN (před započetím práce) tento modul neexistoval a některé z jeho předpokládaných funkcí vykonávaly ostatní moduly.

- Code Repository – Základní funkce jsou uložení balíčku s kódem a poskytnutí obsahu balíčku ostatním modulům (Scheduler a Code Distribution). Balíčky jsou v uzlu SAN ukládány ve formě souborů a jsou označeny názvem aplikace, identifikátorem aplikace a příslušnou příponou.

- Kernel – Inicializuje uzel aktivní sítě SAN a zodpovídá za propojení jeho jednotlivých modulů. Poskytuje služby jako odeslání kapsle a spuštění aplikace (ve spolupráci s ostatními moduly).

- Routing Table – Modul slouží k získávání směrovacích informací pro posílání kapslí. Směrovacími informacemi se rozumí adresa rozhraní následujícího uzlu v cestě k cílovému uzlu a adresa výstupního rozhraní na tomto uzlu.

- Security Manager – Modul má za úkol poskytovat služby spojené se zabezpečením uzlu, od přístupu uživatelů k uzlu až po kontrolování vykonávání jednotlivých aplikací.

- Soft State – Slouží aplikacím, běžícím na uzlu, k uchování dat na přechodnou dobu. Spolupracuje pouze s modulem Application Space.

- Incoming Queue – Tvoří místo pro všechny příchozí kapsle z Network Interfaces. Odsud jsou předány do modulu Code Manager v případě kapslí s aplikacemi nebo modulu Scheduler, pokud se jedná o datové kapsle.

- Outgoing Queue – Slouží jako úložiště kapslí, které odchází z uzlu. O přidání kapslí do Outgoing Queue se stará modul Kernel. Na základě adres obsažených v kapsli jsou pak odesílány příslušnými síťovými rozhraními na sousední uzly.

- Network Interfaces – Tento modul odděluje samotný uzel aktivní sítě od implementace sítě. V projektu SAN je v současné době sítí použitou pro komunikaci mezi uzly klasická IP síť. Ostatní moduly jsou díky této vrstvě nezávislé na typu sítě.

- Application Space – Tento modul poskytuje prostor pro běh aplikací. Aplikace mohou využívat předem definovaných služeb uzlu. Mezi tyto služby patří především: spuštění další aplikace; čekání na ukončení jiné aplikace; získání konzolového vstupu a výstupu; přístup k Soft State; odeslání kapsle.

- Capsule Space – Obdoba Application Space, ovšem poskytuje služby uzlu běžícím kapslím. Služby: identifikace uzlu, na kterém se kapsle nachází; odeslání další kapsle; předání dat aplikaci příslušející ke kapsli; získání identifikace zdrojového a cílového uzlu kapsle; získání identifikace aplikace, pro kterou je kapsle určena; přístup k Soft State; ukončení dalšího vykonávání této kapsle v aktivní síti.

Z obr. 2 je dobře patrné, že modul distribuce kódu je jedním z klíčových prvků celého uzlu.

4.4.2. Změny v modulech nutné k implementaci distribuce kódu

- Scheduler – Žádné zásadní změny nebudou nutné.
- Interpret – Žádné zásadní změny nebudou nutné.
- Code Repository – Modul, který si vyžádal nejrozsáhlejší změny. Je nutné zajistit možnost odstranění aplikace z úložiště při příchodu nové aplikace z důvodu omezení velikosti úložiště a zároveň uchovávat aplikaci po dobu, kdy je nutně potřeba. Po příchodu datové kapsle, jejíž kód je třeba vykonat, na uzel, je nejprve zjištěno, zda je potřebná aplikace v úložišti přítomna a kapsle je zařazena do fronty kapslí čekajících na kód, nebo přímo do fronty kapslí čekajících na vykonání. V obou případech musí modul Code Repository zajistit, aby aplikace nebyla odstraněna mezi tím, kdy byla její přítomnost potvrzena a vykonáním kapsle. Dále je nutné upravit stávající strukturu Code Repository a způsob uložení aplikací tak, aby byla zajištěna bezpečnost z hlediska současného přístupu ostatních modulů k úložišti.
- Kernel – Zde je nutné přehodnotit rozsah zodpovědnosti a pravomocí modulu, protože některé z nich budou přesunuty do nově vytvořeného modulu Code Distribution.
- Routing Table – Žádné zásadní změny nebudou nutné.
- Security Manager – Žádné zásadní změny nebudou nutné.
- Soft State – Žádné zásadní změny nebudou nutné.
- Incoming Queue – O akcích provedených s příchozími kapslemi nově rozhoduje modul Code Distribution. Bude možné kapsli přímo odeslat na následující uzel na její cestě, bez nutnosti ukládat ji do fronty k vykonání na uzlu (pokud se jedná o kapsli žádosti o aplikaci a aplikace se na uzlu nenachází).
- Outgoing Queue – Do Outgoing Queue bude moci kapsle přidávat pouze modul Kernel. Tím bude zajištěna vyšší kontrola nad odesílanými kapslemi.

Zdrojem kapslí budou moduly Kernel (kapsle vzniklé za běhu aplikací), Code Distribution (kapsle s žádostí o aplikace a kapsle obsahující aplikace jako odpověď na žádost) a modul Interpret (kapsle představující výsledek běhu jiné kapsle).

- Network Interfaces – Žádné zásadní změny nebudou nutné.
- Application Space – Žádné zásadní změny nebudou nutné.
- Capsule Space – Žádné zásadní změny nebudou nutné.

Nové moduly:

- Code Distribution – Modul bude obsahovat snadno zaměnitelnou část nazvanou Strategie distribuce kódu a obecnou část pro podporu těchto strategií.
- Traffic Logger – Cílem modulu je shromažďování a zpřístupnění informací o kapslích, které uzlem prochází. Konkrétně se bude jednat o tyto sledované hodnoty:
  - Počet příchozích a odeslaných kapslí jednotlivých typů (datové kapsle, kapsle přenášející aplikace a kapsle s žádostí o aplikaci) rozdělený podle uzlů, ze kterých kapsle přišli, resp. na které směřovaly.
  - Identifikace uzlů, ze kterých přicházejí kapsle s aplikacemi a na které byly kapsle s aplikacemi odeslány, rozdělené podle aplikací.

  Data budou použita k rozhodování strategie distribuce kódu a také pro měření její efektivity a budou zaznamenávána pro každou prošlou kapsli.

4.4.3. Kapsle

Jak již bylo popsáno v kapitole 2, aktivní síť funguje na principu přenosu kapslí, které tvoří obdobu paketů z tradičních IP sítí. Každá kapsle obsahuje:

- Identifikátor aplikace (balíku kódu), který k ní přísluší.
- Adresy čtyř síťových rozhraní: zdrojového, cílového a předchozího spolu s následujícím (vzhledem k současné poloze kapsle).
- Indikátor, zda se má kód kapsle spustit na každém uzlu, kterým kapsle prochází, nebo pouze na cílovém. Tato funkčnost bude v práci nově implementována, původní stav je vykonání kapsle na každém z navštívených uzlů.
- Samotná data. V prostoru pro data lze přenášet místo dat i balík kódu a tím ho distribuovat.

V projektu SAN existují tři typy kapslí:

- Datová – Kapsle přenášející data a na spustitelný kód se odkazují pomocí jeho identifikátoru.
- Přenášející aplikaci – Slouží k distribuci aplikací mezi uzly aktivní sítě. Kapsle s aplikací nemůže obsahovat data určená pro tuto aplikaci (a naopak, datová kapsle neobsahuje aktivní kód). Je součástí protokolu distribuce kódu.
- Žádost o aplikaci – Kapsle neobsahující žádná data ani aplikaci, pouze je nastaven příznak a identifikátor aplikace, ke které kapsle přísluší, určuje, o jakou aplikaci je žádáno. Je součástí protokolu distribuce kódu.

4.4.4. Code Distribution

Modul Code Distribution se do procesu zpracování kapsle na uzlu zapojí hned na počátku po jejím přijetí. Jeho pravomocí bude rozhodnout, zda kapsli umístit do Incoming Queue a tím ji zpřístupnit zbytku uzlu, či nikoli. Poté, co kapsle bude (nebo nebude) předána do Incoming Queue, bude znovu poskytnuta modulu Code Distribution. Jeho zodpovědností bude se o kapsli postarat, pokud nebyla přidána k dalšímu zpracování.

Modul bude také poskytovat službu nalezení aplikace, resp. odeslání kapsle s žádostí o aplikaci. S tím souvisí nalezení nejvhodnějšího uzlu, na kterém je s největší pravděpodobností aplikace uložena.

Další funkcí tohoto modulu bude odeslání aplikace uzlu na základě kapsle s žádostí o aplikaci. Zde se nemusí jednat o doslovné odeslání aplikace, ale také o pokračování v hledání této aplikace na dalších uzlech.

4.4.5. Strategie

Nedílnou součástí modulu Code Distribution bude již zmiňovaná strategie (viz kapitolu 4.4.1). Strategie bude vytvořena jako snadno zaměnitelná, s jasně definovaným rozhraním a bude to právě tato část, která bude provádět všechna rozhodnutí. Proto, aby bylo možné měřit výsledky strategie, budou vytvořeny dvě různé implementace, jejichž výsledky budou navzájem porovnány.

První strategie bude simulovat původní implementaci distribuce kódu, samozřejmě s přihlédnutím k tomu, že její vlastnosti nelze naprosto přesně kopírovat, protože se změní vlastnosti mnoha modulů tvořících uzel SAN. Druhá strategie bude vytvořena na základě provedené analýzy a bude plně využívat všech nových možností zpřístupněných nově implementovanou funkčností.

Strategie č. 1 – původní

Tato strategie kopíruje původní způsob distribuce kódu. Přidáno je rozhodování pomocí modulu Traffic Logger (v případě nutnosti poslat žádost o aplikaci jako reakci na jinou předchozí žádost), který nebyl předtím k dispozici. Pokud by nebyl použit, hrozilo by, že některé žádosti o aplikace by nebyly obslouženy vůbec a tyto aplikace by pak neukončily svůj běh korektně. Z důvodu testování je lepší se tomuto nebezpečí vyhnout, protože podobná chyba v distribuci kódu se špatně signalizuje a je těžké ji rozeznat od mnohem vážnějších chyb, se kterými není předem počítáno. Následuje popis chování této strategie ve čtyřech základních situacích, které strategie musí řešit.

Kapsle předané uzlu ke zpracování:

Kapsle bude předána modulu Incoming Queue, pouze pokud je tento uzel jejím cílovým uzlem anebo pokud se nejedná o cílový uzel, ale kód kapsle se má vykonat na každém uzlu, kterým prochází a jedná se o datovou kapsli.

Zpracování kapsle samotnou strategií:

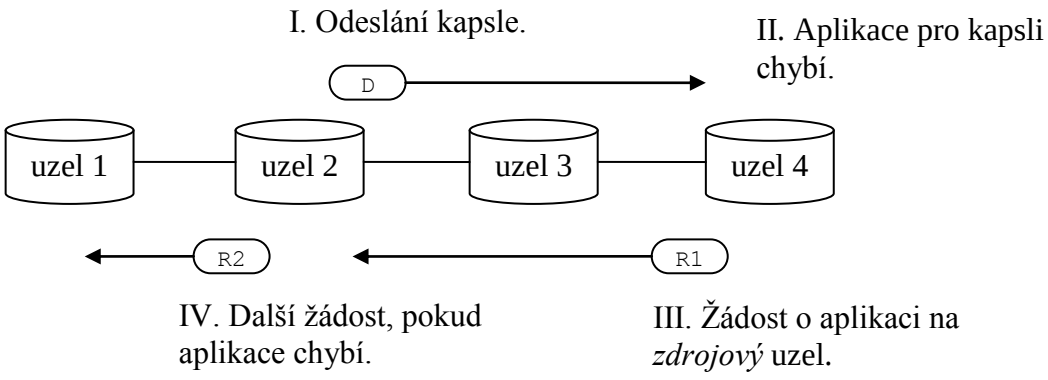
Pokud je kapsle předána modulu Incoming Queue, strategie s ní již nic dále neprovádí. V případě, že se jedná o žádost o aplikaci, která směřuje na tento uzel, je žádost obsloužena (viz oddíl Odpověď na žádost o aplikaci). V ostatních případech je kapsle poslána dále svým směrem.

Vyhledání uzlu s aplikací:

Uzel, na který bude poslána žádost o aplikaci v případě, že se na tomto uzlu požadovaná aplikace nenachází, je jednoduše uzel, ze kterého přišla kapsle, která způsobila potřebu aplikaci získat.

Odpověď na žádost o aplikaci:

Pokud se na uzlu aplikace nachází, je odeslána kapslí na uzel, který o ni požádal. V opačném případě je odeslána kapsle s žádostí o aplikaci na uzel, který je určen modulem Traffic Logger jako nejvhodnější.



Obr. 3.: Princip původní strategie distribuce kódu

Na obr. 3 je zobrazeno odeslání datové kapsle (označené písmenem D) z uzlu 2 na uzel 4, na kterém se ovšem požadovaná aplikace nenachází (kroky I. a II.). V aktivní síti je použita původní strategie distribuce kódu a proto uzel 4 posílá žádost o aplikaci (R1) na uzel 2 (krok III.). Ten, pokud aplikaci nemá k dispozici, může pokračovat odesláním další žádosti (R2, krok IV.).

Strategie č. 2 – nově vytvořená

Strategie je výsledkem analýzy popsané výše. Následuje popis chování strategie v jednotlivých situacích.

Kapsle předané uzlu ke zpracování:

Modulu Incoming Queue (a tím i celému uzlu) budou předány datové kapsle, které se mají vykonávat na každém uzlu. Stejně tak všechny kapsle, které přenáší aplikace (aby aplikace mohla být uložena do úložiště) a kapsle, pro něž je tento uzel jejich cílový. Každý požadavek na aplikaci bude ošetřen přímo strategií distribuce kódu a nebude předán uzlu.

Zpracování kapsle samotnou strategií:

Kapsle s aplikací, které nejsou určené pro tento uzel, a přesto jsou předány modulu Incomming Queue, jsou přeposlány dál směrem k cílovému uzlu. Žádosti o aplikaci jsou obslouženy v modulu této strategie (viz níže), ostatní kapsle jsou také přeposlány dál směrem k cílovému uzlu.

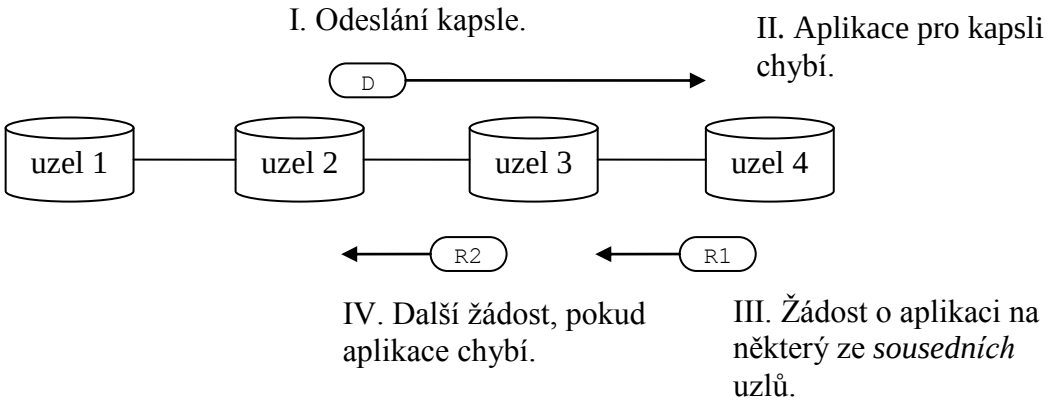
Vyhledání uzlu s aplikací:

Žádost o aplikaci je poslána sousednímu uzlu, ze kterého kapsle s daty přišla. Namísto zdrojového uzlu u předchozí strategie se jedná o předchozí uzel na cestě kapsle na tento uzel.

Odpověď na žádost o aplikaci:

Aplikace je odeslána přímo z tohoto uzlu pokud se nachází v úložišti. V opačném případě je vytvořena nová žádost o aplikaci a jako její zdrojový uzel je uveden

původní zdrojový uzel předchozí žádosti (aplikace se tak nebude posílat na tento uzel, ale skutečně na ten, který o aplikaci původně žádal). Cíl nové žádosti se vybere na základě dat o kapslích se stejnou aplikací (nebo alespoň kapslích s daty určených pro aplikaci), které již dříve prošly uzlem.



Obr. 4.: Princip nově vytvořené strategie distribuce kódu

Na obr. 4 je zobrazeno odeslání datové kapsle (označené písmenem D) z uzlu 2 na uzel 4, na kterém se ovšem požadovaná aplikace nenachází (kroky I. a II.). V aktivní síti je použita nová strategie distribuce kódu a proto uzel 4 posílá žádost o aplikaci (R1) na sousední uzel 3 (krok III.). Ten, pokud aplikaci nemá k dispozici, může pokračovat odesláním další žádosti (R2, krok IV.). Nová strategie je spekulativní (na uzlu 3 kód být nemusí), přesto dosahuje lepších výsledků, než původní strategie (viz kapitolu 6).

4.5. Měření výsledků

Výsledkem práce by měla být spolehlivá a efektivní metoda distribuce kódu. K zhodnocení těchto parametrů bude nutné vytvořit testovací prostředí (aktivní síť) a aplikace, které se budou v síti distribuovat. Zároveň bude nutné přizpůsobit výstupy aplikací a i samotného uzlu tak, aby bylo možné sledovat parametry důležité pro kontrolu a hodnocení mechanismu distribuce kódu.

Hlavním sledovaným parametrem bude počet kapslí s žádostí o aplikaci, které nemohly být obslouženy na uzlu a museli být přeposlány na jiný uzel. Tento počet bude sledován, protože ukazuje, kolikrát byla žádost o aplikaci poslána na uzel, na kterém se aplikace nenachází a tím pádem zbytečně. Jako další se bude samozřejmě sledovat počet samotných odeslaných aplikací.

Výsledky práce budou porovnány se stavem před její implementací, kdy byla v uzlu SAN distribuce kódu řešena velmi jednoduchým, přímočarým a v důsledku nespolehlivým způsobem (např. pokud žádost o aplikaci neuspěla na cílovém uzlu, v původní strategii by již nenásledovaly žádné další kroky a požadovaná aplikace by nikdy nebyla nalezena). Ten simuluje implementace strategie distribuce kódu označovaná jako původní.

4.6. Oblasti, které práce nezahrnuje

Vybrané problémy související s touto problematikou, jenž nejsou ještě v projektu SAN plně vyřešeny, distribuci kódu ovlivňují, ale na které se práce nevztahuje.

Práce neřeší nalezení aplikace, pokud je spuštěna přímo samotným uživatelem. V takovém případě se předpokládá, že uživatel bude moci aplikaci na uzel před jejím spuštěním nainstalovat. Práce se zaměřuje se pouze na požadavky na spuštění aplikací v důsledku příchodu kapsle na uzel.

V této práci je kladen důraz především na spolehlivost a efektivitu protokolu distribuce kódu. Bezpečnost bude řešena až s celkovou bezpečnostní koncepcí uzlu aktivní sítě SAN v rámci dalších projektů.

5. Řešení (Programátorská dokumentace)

Tato kapitola popisuje samotnou implementaci práce, která vychází z analýzy v kapitole 4. Jsou zde popsány konkrétní algoritmy a prostředky použité při implementaci. Pro kód, který je distribuován mezi uzly a tvoří aplikace a kód kapslí jsou dále používána dvě označení: balík kódu a aplikace.

5.1. Implementace uzlu SAN

Distribuce kódu je implementována v rámci projektu SAN. Pro celkový teoretický přehled viz kapitolu 4. Zde je uveden souhrn informací o implementaci uzlu SAN důležitých z pohledu programátora pro pochopení dalších kapitol (ale také vhodný např. při případném rozšiřování a vytváření nové funkčnosti uzlu).

Celý projekt je od začátku programovaný v jazyce Java. Tím jsou daná jasná pravidla, možnosti i omezení.

Detailnější pohled na implementaci modulů popsaných teoreticky v kapitole 4.4:

- Scheduler – Balík `san.core`, třída `Scheduler`. Ve třech vláknech se stará o pořadí a načasování spuštění aplikací a kapslí. Jmenovitě jde o vlákno `applicationsSchedulerThread` pro aplikace, které se nachází ve frontě `waitingApplications` ve formě instancí třídy `Process`. Druhým vláknem je `capsulesSchedulerThread` starající se o plánování kapslí z fronty `waitingCapsules`. Obě tato vlákna pro kapsle i aplikace vytváří instance tříd implementujících rozhraní `IInterpreter`, které představují interpret daného kódu a umožňují jeho interpretaci sledovat a ovládat. Posledním vláknem je `capsulesPreparationThread`, které čeká na příchozí kapsle z fronty `IncomingQueue`. V případě, že pro kapsle je v úložišti `CodeRepository` příslušná aplikace, vytvoří instanci `ProcessCapsule`, přiřadí ji danou kapsli (instanci `TransmitCapsule`) a aplikaci (ve formě `CodePackage`) a zařadí tento objekt do fronty `waitingCapsules`. V případě, že pro kapsli není aplikace přítomna, je přidána do fronty `capsulesWaitingForCode` a následuje volání metody

`Repository.tryToSendNextRequestForApp()`, které má za úkol poslat další požadavek na aplikaci (požadavky se řadí ve frontě).

- Interpret – Balík `san.interpreter`, třída `InterpretManager`, rozhraní `IInterpreter` a třídy které jej implementují. `InterpretManager` má za úkol pro daný `CodePackage` vytvořit instanci interpretru, která bude umožňovat kód, představovaný instancí `CodePackage`, interpretovat. V současné době jsou v projektu SAN přítomny dva interpretry, jeden pro balíky kódu typu BeanShell a druhý pro balíky typu SAN Package. Více o těchto balících v popisu modulu Code Repository. Řízení běhu kódu má na starosti modul Scheduler.

- Code Repository – Balík `san.applicationRepository`, třídy `Repository`, `CodePackage`, `CodePackageSymLink`, `SanCodePkg`, `BshCodePkg` a rozhraní IRepository. Tento modul byl v průběhu implementace práce výrazně změněn, zde jsou proto uvedeny jen obecné informace, které i po vytvoření modulu Code Distribution a s ním souvisejících změn zůstaly v platnosti. Hlavním posláním modulu je ukládání kódu (aplikací). Každý balík kódu je uložen ve formě souboru na pevném disku. Jméno adresáře s těmito soubory určuje v třídě `Repository` proměnná `repositoryDir`. Soubor s kódem je zip archiv s přesně danou strukturou (podobnou archivu jar). Je reprezentován třídou `CodePackage`. Její dvě podtřídy, `BshCodePackage` a `SanCodePackage`, umožňují přistupovat k obsahu souborů obou výše zmiňovaných druhů balíků (a získat tak jméno balíku, obsah třídy s metodou `main` ve formě instance `InputStream`, obsah dalších tříd u balíku, hodnoty konstant definovaných v manifestu balíku atd.). Pomocnou třídou, která také reprezentuje soubor s balíkem kódu je `CodePackageSymLink`. Tato třída uchovává informace jako čas vytvoření balíku, identifikaci (vytvořenou pomocí hashovaní funkce, reprezentovanou třídou `CodePackageIdent`) a hlavně počet otevření balíku, tzn. kolikrát je v současné době kód z balíku používán k vykonání kapsle nebo aplikace. Balík s kódem se samozřejmě nesmí z úložiště odstranit, pokud je používán.

- Kernel – Balík `san.core`, třída `Kernel`. Třída `Kernel` obsahuje metodu `main` a slouží tak ke spuštění celého uzlu. Parametrem spuštění může být jméno souboru s nastavením, který je detailně popsán v příloze A (pokud není zadán, použije se výchozí hodnota ‚settings.xml'). Po spuštění je vytvořena instance třídy `Kernel`, která ve svém konstruktoru inicializuje všechny ostatní třídy modulů popisovaných v tomto seznamu a zajistí jejich vzájemnou provázanost. Poté je načteno nastavení ze souboru a podle něj jsou vytvořena rozhraní pro komunikaci s ostatními uzly, nastaveny směrovací informace v modulu Routing Table a cesta k adresáři, který slouží jako úložiště kódu. V průběhu inicializace jsou jednotlivé moduly také spuštěny (většina je tvořena samostatnými vlákny) a proto je po skončení konstruktoru třídy `Kernel` celý uzel již běžící a připravený spolupracovat s ostatními uzly v aktivní síti. Během inicializace instance třídy `Kernel` je také vytvořen jednoduchý webový server, který umožňuje přehledným způsobem zobrazovat důležité informace o běhu uzlu (více v kapitole 5.3.2 v části Zobrazení statistiky kapslí a aplikací a v příloze A).

- Routing Table – Balík `san.routing`, třídy `Router`, `RoutingTable`, `RoutingRecord` a rozhraní `IRouting`, `IRoutingTableControl`. Třída `Routingtable` slouží k ukládání směrovacích informací, které se používají při odesílání kapslí. Informace jsou uložené ve tvaru `Map<NetIdentifier, SortedSet<RoutingRecord>>`.
  `NetIdentifier` jednozančně určuje rozhraní cílového uzlu (tvoří ho dvojice 16 bytů dlouhých identifikátorů `Guid`: číslo rozhraní a číslo sítě), `RoutingRecord` potom určuje, jaké je následující rozhraní (a tím i následující uzel) na cestě směrem k cílovému rozhraní a jaké rozhraní lokálního uzlu bude použito jako odchozí (každému záznamu `RoutingRecord` je možné přiřadit váhu, podle jeho vlastností a na základě této váhy jsou potom řazeny a vybírány, pokud k jednou cílovému uzlu existuje více záznamů). Takto jsou uloženy cesty k rozhraním ostatních uzlů načtené ze souboru s nastavením. Instance třídy `Router` obsahuje instanci `RoutingTable` a kromě toho umožňuje získat výchozí cestu z uzlu

(uložená také v souboru s nastavením), která se používá v případě, že o požadovaném směru nejsou v `RoutingTable` uložené žádné informace. Třída `Router` také poskytuje informace o rozhraních ostatních uzlů, která jsou přímo spojená s rozhraními tohoto uzlu.

- Soft State – Balík `san.softstate`, třídy `SoftState` a `TimedObject`. Třída `Softstate` umožňuje aplikacím pomocí svých metod ukládat na omezenou dobu libovolné objekty. Ty jsou uloženy v instancích třídy `TimedObject`, která ke každému objektu vyžaduje také zadání času, po který je objekt platný. Uložené objekty jsou pravidelně kontrolovány a odstraňovány.

- Incoming Queue – Balík `san.incomingQueue`, třída `IncomingQueue`, rozhraní `IPastReceiving`. Třída `IncomingQueue` představuje místo pro příchozí kapsle. Implementuje rozhraní `IPastRecieving`, které jí umožňuje se registrovat jako posluchač u tříd, které dědí od `NetListener` a upozorňují na příchozí kapsle. Tyto třídy jsou přímo zodpovědné za spojení místního síťového rozhraní s rozhraním na sousedním uzlu a starají se o příjem kapslí (jednomu lokálnímu síťovému rozhraní odpovídá jedna instance `NetLister`). Každá kapsle je reprezentována objektem třídy `TransmitCapsule`. Ten obsahuje především samotná data, přenášená kapslí (instance třídy `TransmitData`, v podstatě pole bytů). Dále pak `TransmitCapsule` obsahuje:

  - identifikaci čtyř síťových rohraní (`NetIdentifier`): zdrojové (atribut `src`), cílové (`dest`), předchozí (`prev`) a následující (`next`)
  - atribut `hash` typu `CodePkgIdent`, který určuje, k jaké aplikaci kapsle přísluší
  - `boolean` atributy určující, zda jde o kapsli přenášející aplikaci nebo je žádostí o aplikaci (pokud mají oba hodnotu `false`, jedná se o kapsli přenášející data) a zda se má kód kapsle vykonat na všech uzlech, kterými prochází, nebo pouze na cílovém (atribut `executeContinuously`).

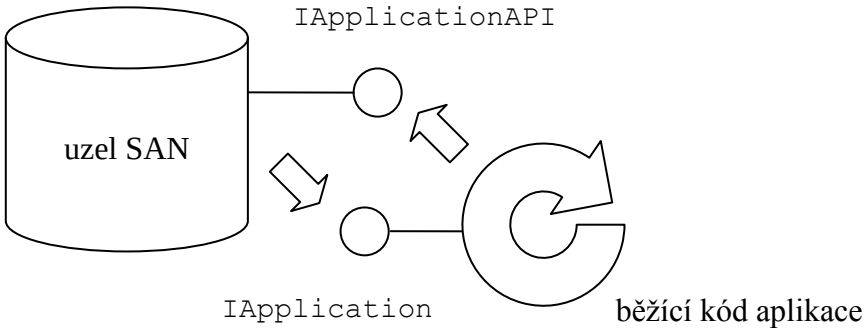
Poté, co některé ze síťových rozhraní přijme kapsli, je předána `IncomingQueue`. Zde může být zařazena do jedné z front `dataCapsules` a `programCapsules` (obě typu `Queue<TransmitCapsule>`), podle jejího obsahu. O tom rozhoduje metoda `shouldCapsuleBeStoredInQueue` ze třídy, která dědí od `CodeDistributionStrategy` (kapsle s žádostí o aplikaci jsou obslouženy přímo v `CodeDistributionStrategy` a nejsou ukládány do fronty). Po zařazení kapsle do fronty je opět předána modulu Code Distribution k dalšímu možnému zpracování. Z obou front jsou pak kapsle vyzvedávány modulem Scheduler.

- Outgoing Queue – Balík `san.outgoingQueue`, třída `OutgoingQueue`. Jak je patrné z názvu, jedná se o opak `IncommingQueue`. Instance třídy `OutgoingQueue` slouží ke shromažďování kapslí, které mají být odeslány z uzlu. Odesílání se děje pomocí instancí třídy `NetOut`, která představuje spojení mezi lokálním síťovým rozhraním a rozhraním na sousedním uzlu, podobně jako `NetListener`, ovšem v případě `NetOut` se jedná o odesílání kapslí a tím pádem o spojení směrem z lokálního uzlu na sousední. Kapsle nejsou nijak rozdělovány podle typu, všechny jsou odesílány stejným způsobem (serializace objektu `TransmitCapsule` a jeho poslání skrz socket).

- Network Interfaces – Balík `san.networkInterfaces`, nejdůležitější třídy: `ConnectionSolver`, `ConnectionSolverManager`, `IncomingPacket`, `NetInterface`, `NetOut`, `RemoteInterface`, `TcpConnection`, `TcpListener`, `TcpOut`, `UdpConnection`, `UdpListener`, `UdpOut`, nejdůležitější rozhraní: `IPastRecieving`. Instance těchto tříd jsou použity pro vytvoření a obsluhu spojení mezi uzly. Na začátku vytváření spojení je metoda `findNeighbours()` ze třídy `NetInterface`, která je periodicky volána. V důsledku toho je odesílán inicializační packet na multicast adresu (a port) uvedenou v souboru s nastavením u daného síťového rozhraní. Packet obsahuje hlavičku (obecné informace), identifikaci lokálního síťového rozhraní (číslo rozhraní a číslo sítě), lokální multicast port a metodu spojení, kterou je rozhraní schopno akceptovat (TCP nebo UDP). Při správné konfiguraci na stejné multicast adrese poslouchá také síťové rozhraní vedlejšího uzlu a po výměně inicializačních paketů jsou pro spojení lokálního a vzdáleného síťového rozhraní vytvořeny instance `ConnectionSolverManager`, `NetLister` a `NetOutfp`.

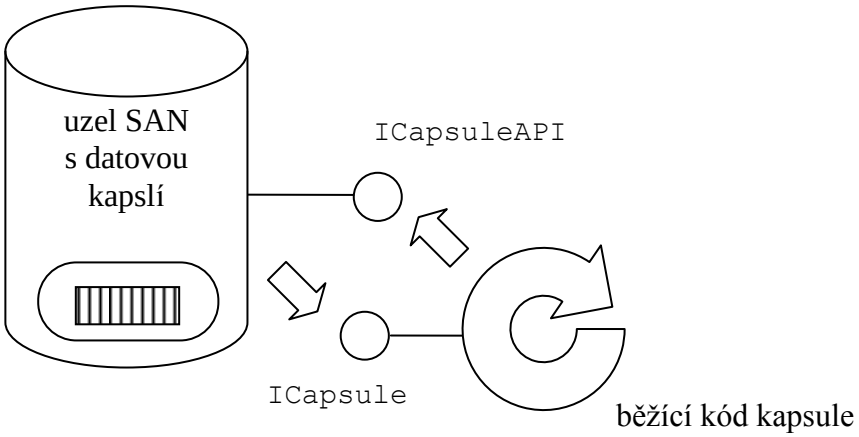
- Application Space – Balíky `san.application` a `san.core`, rozhraní `IApplication`, `IApplicationAPI`, třída `ProcessApplication`. Rozhraní `IApplication` musí být implementováno každou aplikací, která je určena pro aktivní síť uzlů SAN. Definuje jen jedinou metodu, `main`, která umožňuje spuštění aplikace. Při spuštění je pak aplikaci předán jako parametr objekt typu `IApplicationAPI`, který slouží aplikaci jako přístupový bod pro využívání služeb uzlu. Mezi základní služby patří předání dat aplikaci pomocí mechanismu registrování aplikace jako posluchače u uzlu metodou `addOnSendDataListener`. Tím je umožněno např. poslání dat z kapsle, která dorazila na uzel, příslušné aplikaci ke zpracování. Dalšími službami v rozhraní `IApplicationAPI` jsou odeslání kapsle (metoda `injectCapsule`), přístup k modulu Soft State (metoda `getSoftState`), čekání na skončení jiných aplikací (`waitForApplication`), nebo přístup ke konzolovému vstupu a výstupu uzlu (`getStdIn`, `getStdOut`). Rozhraní

`IApplicationAPI` je implementováno třídou `ProcessApplication`, která představuje jednu běžící aplikaci na uzlu.



Obr. 5.: Komunikace mezi uzlem a běžící aplikací

- Capsule Space – Balíky `san.application` a `san.core`, rozhraní `ICapsule`, `ICapsuleAPI`, třída `ProcessCapsule`. Rozhraní `ICapsule` je implementováno každou třídou představující kód kapsle. Tento kód se nepřenáší v datových kapslích v aktivní síti, ale je distribuován stejně jako aplikace ve zvláštních kapslích z uzlu na uzel. Datové kapsle obsahují pouze data a identifikaci aplikace. Pro každou kapsli je pak na uzlu (pokud se na něm má vykonat) vytvořena instance příslušné třídy implementující `ICapsule`. Tomuto objektu je jako parametr předána instance třídy `PorcessCapsule`, která implementuje `ICapsuleAPI` a umožňuje kapsli přistupovat k službám uzlu. Mezi ně patří: přístup k datům kapsle (metoda `getDataLoad`), získání identifikátoru zdrojového rozhraní (`getSource`), získání a nastavení cílového rozhraní kapsle, oznámení o ukončení cesty kapsle (kapsle nemá bát posílána dále, metoda `finished`), odeslání nové kapsle a předání dat aplikaci.



Obr. 6.: Komunikace mezi uzlem a běžícím kódem kapsle

5.2. Požadavky na implementaci distribuce kódu

Distribuce kódu je jedním z modulů uzlu SAN, proto je přirozeně naprogramovaná v jazyce Java. Kromě nalezení algoritmu distribuce kódu, je samozřejmě požadavkem i jeho efektivní implementace.

5.3. Nové a upravené moduly uzlu SAN

Zde je popsána implementace modulů nově vytvořených, anebo změněných, během vytváření protokolu pro distribuci kódu.

5.3.1. Úložiště kódu

V kapitole 5.1 byla popsány hlavní principy ukládání kódu na uzlu (modul Code Repository). Způsoby uložení kódu se během implementace nezměnily (kód je uložen v souboru na pevném disku, v uzlu SAN je reprezentován instancemi třídy `CodePackage`, která dále každá obsahuje instanci `CodePackageSymlink`). Změnily se ovšem způsoby zacházení s takto uloženým kódem. Ty jsou implementovány ve třídě `Repository` z balíku `san.applicationRepository`

Hlavní změnou je možnost balík kódu z úložiště odstranit. Děje se tak automaticky, pokud hrozí, že bude překročena nastavená maximální kapacita úložiště. Uložený kód ale musí být ochráněn před smazáním po celou dobu, po kterou je používán. Hlavním prostředkem zde je celočíselná proměnná `numberOfOpens` ve třídě `CodePackageSymLink`, která určuje, kolikrát je daný balík v současné době používán (může běžet více instancí stejné aplikace nebo kapsle současně). Přirozeně nejsou odstraňovány balíky, které mají hodnotu této proměnné větší než nula. S balíky kódu pracují třídy `Scheduler` (spuštění kódu kapsle) a `Kernel` (spuštění aplikace uživatelem nebo jinou aplikaci). Přistupují k nim pomocí metod `getCodePkg` s parametrem jméno aplikace (řetězec) nebo identifikátor aplikace (typu `CodePkgIdent`). Právě v těchto metodách je zvyšováno číslo `numberOfOpens` u instancí `CodePackageSymLink` odpovídajícího balíku kódu. Toto číslo se přirozeně snižuje po ukončení vykonávání kódu aplikace nebo kapsle. K indikaci ukončení vykonávání slouží metoda

`dontProtectAppFromDeleteAnyLonger` třídy `Repository`, která snižuje právě hodnotu proměnné `numberOfOpens`.

Ovšem balík kódu nesmí být odstraněn ani v případě, kdy je dopraven kapslí na uzel, je uložen do úložiště a interpretace kódu kapsle, které ho vyžádala, ještě nezačala (kapsle jsou řazeny do fronty, kde čekají na interpretaci a počet současně běžících kapslí je samozřejmě také omezen) a proto je proměnná `numberOfOpens` stále na své výchozí nulové hodnotě. K ochraně nově příchozích balíků kódu slouží proměnná `protectedApps` (typu `Map<CodePkgIdent, Boolean>`). Do ní je pro každý balík kódu uložena hodnota `true` v momentě, kdy je odesílána žádost o aplikaci (voláním metody `sendRequestForApp` ze třídy `CodeDistributionStrategy`). V metodách `getCodePkg` je pak nejen inkrementována hodnota `numberOfOpens` o jedničku, ale také je uložena do `protectedApps` pro požadovanou aplikaci hodnota `false`. Pokud je od tohoto okamžiku do ukončení běhu kódu aplikace (resp. kapsle) požadován stejný balík kódu znovu (protože např. na uzel dorazila další kapsle určená stejné aplikaci), je hodnota `numberOfOpens` inkrementována a aplikace nebude smazána do ukončení běhu druhé kapsle.

Stejným způsobem je uložena hodnota `true` do `protectedApps` pro balíky kódu, které byly v adresáři, sloužícím jako úložiště, při spuštění uzlu. Současně je jim nastavena hodnota `numberOfOpens` na jedna přesto, že nebyly ještě ani jednou spuštěny a tím je zajištěno, že nebudou z úložiště nikdy odstraněny (hodnota `numberOfOpens` nikdy neklesne na nulu). To je důležité proto, aby aplikace byly vždy přítomné alespoň na těch uzlech, na kterých byly nainstalovány a bylo tak zajištěno, že z aktivní sítě nikdy úplně nezmizí.

Odstraňování aplikací je prováděno metodou `clearRepository`. Ta je volána vždy, když počet uložených balíků kódu dosáhne maximálního možného počtu. Metoda ze všech uložených aplikací odstraňuje pouze takové, které nejsou v současnosti používané (`numberOfOpens` je roven nule) a zároveň nejsou chráněny před odstraněním pomocí `protectedApps`. Aplikace je odstraněna jak fyzicky

z adresáře úložiště, tak i všechny informace uložené v objektech `CodePackage`, `CodePackageSymLink` apod.

Pokud modul `Scheduler` dostane ke zpracování kapsli, pro kterou není příslušný kód v úložišti, zavolá metodu `tryToSendNextRequestForApp` třídy `Repository`. Té předá jako parametr kapsli, pro niž není k dispozici kód. V metodě se nejdříve zkontroluje, zda před jejím voláním již kód náhodou nedorazil (díky předchozí žádosti). Pokud kód skutečně není k dispozici, je zkontrolováno, zda neexistuje žádost o tento balík kódu, která byla odeslána, ale na kód se stále čeká. Pokud ne, je kapsle zařazena do fronty. Z této fronty je odebrána až po zavolání metody `tryToSendNextRequestForApp`, která, pokud je v úložišti místo na další aplikaci, postupně odesílá žádosti o kód z fronty a to pomocí modulu Code `Distribution`. To znamená, že žádosti o distribuci kódu nejsou odesílány, dokud není jisté, že pro aplikaci je místo v úložišti.

5.3.2. Záznam pohybu kapslí

Tento modul tvoří tři třídy `CapsuleTrafficLogger`, `CapsuleTrafficLoggerData` a `AppPlacementInfo` v balíku `san.codeDistribution`. Třída `CapsuleTrafficLogger` implementuje dvě rozhraní `CodeStoredListener` a `CapsuleAddedInQueueListener`. To umožňuje tříd zaregistrovat instanci třídy jako posluchače u událostí přidání kapsule do front `IncommingQueue` a `OutgoingQueue` (metoda `capsuleAdded`) a také u uložení aplikace do úložiště (metoda `codeStored`).

U každé kapsle, která přijde na uzel nebo z něj odejde, je uložena informace do instance `AppPlacementInfo`, a ta je poté sama ukládána podle identifikace aplikace příslušející ke kapsli do mapy `appPlacementLog` (typ `Map<CodePkgIdent, AppPlacementInfo>`). Jedné aplikaci je tedy přiřazena jedna instance třídy `AppPlacementInfo`. `AppPlacementInfo` obsahuje čtyři množiny (typu `Set`) identifikátorů síťových rozhraní. Tyto množiny obsahují síťová rozhraní, z nichž byla na uzel poslána kapsle s daty (množina `nodesDataCameFrom`) nebo samotnou aplikací (`nodesAppCameFrom`) a síťová

rozhraní, na něž byly tyto kapsule odeslány (`nodesAppSentTo` a `nodesDataSentTo`). Tato rozhraní jsou ukládána, pokud je kapsle předaná metodě `capsuleAdded` kapslí s daty nebo s aplikací a odstraňována, pokud jde o požadavek na aplikaci.

Instance třídy `CapsuleTrafficLoggerData` slouží k zaznamenávání počtu kapslí prošlých jednotlivými rozhraními uzlu. Jsou proto uloženy v proměnné `capsuleTrafficLog` (typ `Map<NetIdentifier, CapsuleTrafficLoggerData>`), kde `NetIdentifier` představuje identifikátor síťového rozhraní uzlu. Zaznamenáván je počet všech tří typů kapslí a u každého typu je rozlišeno odeslání nové kapsle (vytvořené uzlem) a přeposlání již existující kapsle.

Zobrazení statistiky kapslí a aplikací

Pokud je název souboru s nastavením ve tvaru ‚`settings[celé_číslo].xml`' je ve třídě `Kernel` její při inicializaci vytvořena instance `ControlHttpServer`. Ta slouží jako jednoduchý webový server běžící na adrese ‚http://localhost' a portu číslo `5560 + [celé_číslo_z_názvu_souboru]`. Kód této stránky je vytvářen v metodě `getAppPlacementLogInHtml` třídy `CapsuleTrafficLogger`. Popis obsahu webové stránky je popsán v příloze A.

5.3.3. Strategie distribuce kódu

Balík san.`codeDistribution`.strategies, třídy `CodeDistributionStrategy`, `NaiveCodeDistributionStrategy`, `AskNeighbourDistributionStrategy`. Abstraktní třída `CodeDistributionStrategy` slouží jako předloha pro vytváření strategií distribuce kódu. Každá strategie tedy musí implementovat pět metod:

- `shouldCapsuleBeStoredInQueue` – Metoda je volána v třídě `IncomingQueue` při příchodu kapsle na uzel, návratová hodnota je `boolean` a určuje, zda bude kapsle dále uzlem zpracována, tzn., jestli bude uložena do jedné z front `programCapsules` nebo `dataCapsules` v `IncommingQueue`.

- `dealWithCapsuleAfterQueue` – Je volána opět ve třídě `IncomingQueue`, ihned po rozhodování o zařazení kapsle do fronty pomocí výše popsané metody `shouldCapsuleBeStoredInQueue`. Umožňuje strategii distribuce kódu pracovat dále s kapslí, bez ohledu na to, zda byla zpracována uzlem (konkrétně třídou `Scheduler`).

- `sendApplicationFor` - Není volána v současné době z žádné třídy uzlu, pouze přímo ze tříd dědících od `CodeDistributionStrategy`. Je zamýšlena jako metoda, která uskutečňuje odeslání kapsle s aplikací nebo zajistí nalezení požadované aplikace (odesláním požadavku na aplikaci některému uzlu).

- `sendRequestForApp` – Je volána ze třídy `Repository` v případě, pokud je ve frontě alespoň jedna kapsle, pro kterou je nutné zajistit balík kódu a v úložišti je pro tento kód dostatek místa. Jako parametr volání je předána instance `TransmitCapsule`, která tvoří původní kapsli s daty aplikace, která se v úložišti nenachází. Úkolem metody je odeslat požadavek na aplikaci.

- `getBestAppLocation` – Podobně jako metoda `sendApplicationFor` je i tato volána zatím pouze ze tříd, které dědí od `CodeDistributionStrategy`. Úkolem metody je vrátit identifikátor síťového rozhraní uzlu, na kterém se podle dostupných informací s největší pravděpodobností nachází aplikace příslušející ke kapsli, která je parametrem volání metody.

Implementovány jsou dvě strategie. Jejich chování je popsáno v kapitole 4.4.5. Následuje popis strategií z hlediska implementace:

- `NaiveCodeDistributionStrategy` – Tato strategie simuluje původní chování distribuce kódu v uzlu SAN. Metoda `shouldCapsuleBeStoredInQueue` provádí rozhodování na základě vlastností kapsle samotné a třídy `Kernel`, která umožňuje určit v metodě `isThisNode`, jestli zadané síťové rozhraní patří tomuto uzlu (v tomto

případě cílové síťové rozhraní kapsle). Kapsle s žádostí o aplikaci jsou obslouženy samotnou strategií v metodě `dealWithCapsuleAfterQueue` spolu s kapslemi, které nejsou určen pro tento uzel. Při odesílání aplikace v metodě `sendApplicationFor` se posílá balík kódu s aplikací z úložiště kódu přímo směrem k uzlu, odkud žádost přišla, pokud je k dispozici, nebo je odeslána žádost dál směrem, který určí metoda `getAppPlacementInfo` z `CapsuleTrafficLogger`. Metoda `getBestAppLocation` vrací pro každou kapsli adresu jejího zdrojového síťového rozhraní a proto je nová žádost o aplikaci, vytvořená tímto uzlem, odesílána vždy na zdrojový uzel kapsle.

- `AskNeighbourDistributionStrategy` – Nově implementovaná strategie. V základě je stejná jako předchozí strategie `NaiveCodeDistributionStrategy`, protože principy dané zvolenou architekturou omezují možnosti, jak strategie implementovat. Hlavní odlišností je, jak nakládá s daty, která jsou k dispozici. Metoda `shouldCapsuleBeStoredInQueue` umožňuje uložit do fronty v `IncomingQueue` i kapsle s aplikacemi, které nejsou přímo určené pro tento uzel, a úložiště se díky tomu může poté rozhodnout aplikaci uložit, pokud ji potřebuje. Pokud je nutné odeslat novou žádost o aplikaci, metoda `getBestAppLocation` vrací adresu síťového rozhraní předchozího uzlu v cestě kapsle s daty aplikace, kterou potřebujeme. Při přeposílání žádosti o aplikaci je cílové rozhraní zvoleno jako to, ze kterého již předtím přišla kapsle s aplikací nebo alespoň daty příslušné aplikace (v metodě `sendApplicationFor`).

6. Výsledky práce

V této kapitole jsou obsaženy, popsány a vysvětleny výsledky diplomové práce.

6.1. Aplikace použité k testování

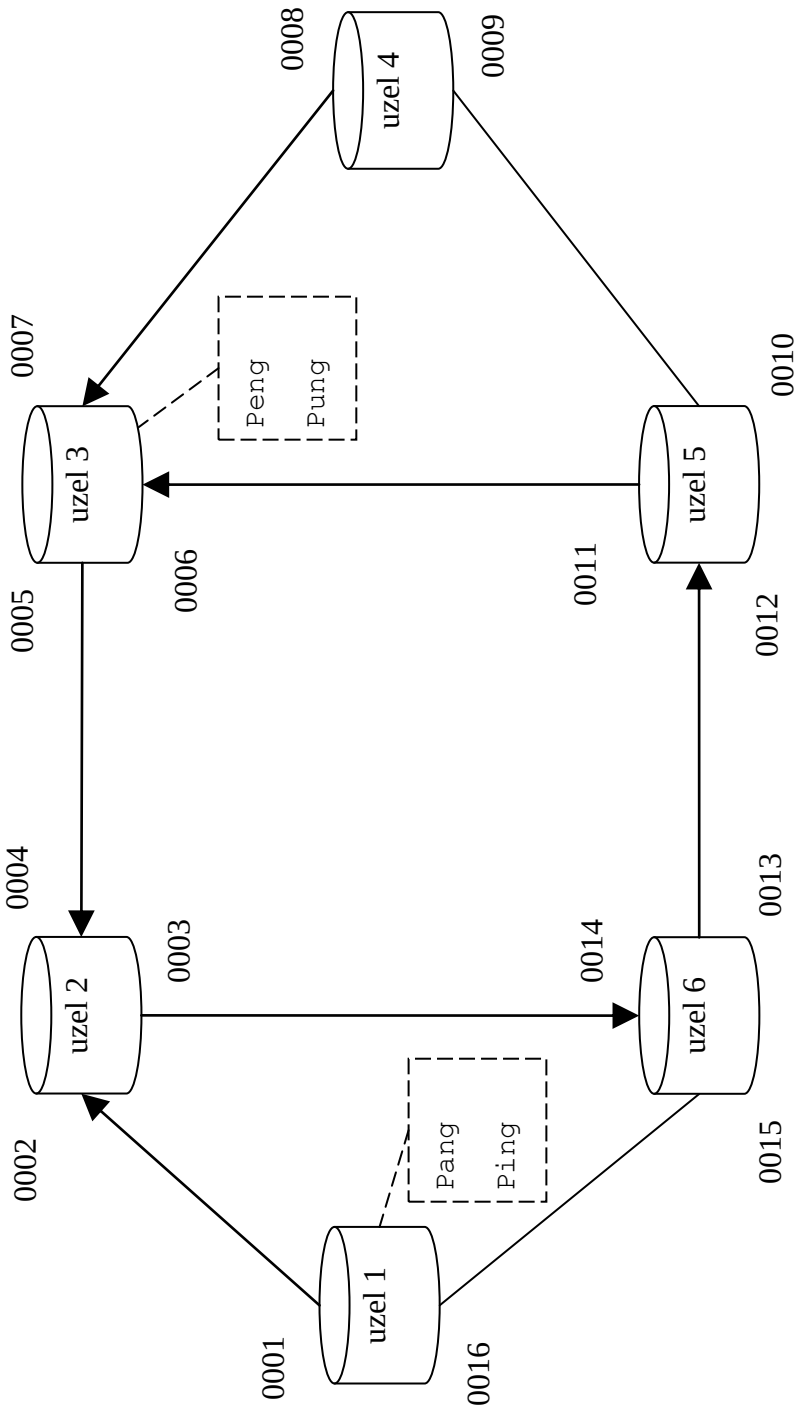
Pro otestování distribuce kódu byly použity čtyři aplikace. Všechny vycházejí ze základní aplikace Ping, která je implementací klasického příkazu ping v prostředí aktivních sítí. Jejím jediným vstupem je adresa cílového uzlu a výsledkem běhu měřit doba přenosu kapsle na cílový uzel a dobu přenosu odpovědi zpět na zdrojový uzel. Z této aplikace byly odvozeny tři další, pojmenované Pang, Peng a Pung. Jejich vlastnosti jsou popsané v tab. 1.

Ping **Pang**	**Peng** **Pung**
Kód kapsle aplikací je vykonán pouze na cílovém uzlu, aplikace se proto nemusí distribuovat do mezilehlých uzlů.	Kód kapsle aplikací je vykonán na všech uzlech, kterými kapsle prochází. Aplikaci je nutné distribuovat do mezilehlých uzlů.

Tab. 1.: Aplikace použité pro testování

Pro automatické testování byla dále vytvořena aplikace RandomRun. Ta slouží k opakovanému spouštění zadaných aplikací. Vstupními parametry jsou: 1) počet opakování spouštění aplikací, 2) jméno první a druhé spouštěné aplikace, 3) doba mezi spuštěními v tisícinách vteřiny, 4) adresy uzlů, které budou předány jako parametry spouštěným aplikacím. Aplikace po spuštění čeká náhodnou dobu v rozmezí 1-2 vteřiny a poté začne spouštět zadané aplikace (musí být přítomné v úložišti uzlu). Spuštěná aplikace je ze zadané dvojice vybrána náhodně. Zde je použit generátor náhodných čísel s rovnoměrným rozdělením. Aplikaci je jako jediný parametr předána jedna z adres zadaných jako poslední parametr při spuštění

RandomRun. Před opakovaným spuštěním další náhodně vybrané aplikace se čeká dobu zadanou jako parametr při spuštění nebo její dvojnásobek (opět náhodně vybráno). Z uvedeného vyplývá, že aplikace RandomRun je vytvořena na míru pro snadné spouštění aplikací Ping, Pang, Peng a Pung.



Legenda			
→	Cesta výchozího směrování z uzlu.		Aplikace na uzlu.
——	Spojení mezi dvěma rozhraními.		
000x	Adresa rozhraní, zkrácený zápis tvaru 00000000-0000-0000-0000-000000000000x		

Obr. 7.: Topologie testovací sítě

6.2. Průběh testování – počet kapslí

Na obr. 7 je zobrazena topologie testovací sítě, která se skládala ze šesti uzlů. Na uzlu č. 1 byly nainstalovány aplikace (popsané detailně v kapitole 6.1) Pang a Ping, na uzlu č. 3 potom Peng a Pung. Spolu s nimi byla na obou uzlech k dispozici i aplikace RandomRun. Ta sloužila ke spuštění těchto dvojic aplikací. Na obou uzlech, č. 1 a č. 3, byl celkový počet spuštění aplikací roven 1000.

Protože byly aplikace spouštěny pomocí RandomRun, byly spouštěny v náhodném pořadí a cílové síťové rozhraní bylo při každém spuštění také vybráno náhodně ze všech rozhraní, které jsou v síti k dispozici (s výjimkou rozhraní na uzlu, kde byla aplikace spouštěna). Na každém uzlu byla kapacita úložiště nastavena na hodnotu dva (tzn. po spuštění uzlu bylo možné do úložiště uložit maximálně dvě další aplikace) a tím byla zajištěna nutnost aplikace v úložištích jednotlivých uzlů obměňovat.

Tento test byl proveden pro obě implementované strategie distribuce kódu. Pro každou z nich byly po proběhnutí testu zaznamenány počty poslaných kapslí jednotlivých typů, rozdělené podle spojení mezi jednotlivými uzly.

6.2.1. Výsledky testování - počet kapslí

zdrojový uzel	cílový uzel	datové kapsle - odeslání	datové kapsle - přeposlání	kapsle s aplikací - odeslání	kapsle s aplikací - přeposlání	žádosti o aplikaci - odeslání	žádosti o aplikaci - přeposlání	celkem datových kapslí - odeslání	celkem datových kapslí - přeposlání	celkem kapslí s aplikací - odeslání	celkem kapslí s aplikací - přeposlání	celkem žádosti o aplikaci - odeslání	celkem žádosti o aplikaci - přeposlání	celkem odesláno	celkem přeposláno
1	1	0	0	0	0	0	0	1169	0	975	0	1	0	2145	0
	2	**599**	**0**	**396**	**0**	**1**	**0**								
	3	0	0	0	0	0	0								
	4	0	0	0	0	0	0								
	5	0	0	0	0	0	0								
	6	**570**	**0**	**579**	**0**	**0**	**0**								
2	**1**	**211**	**527**	**0**	**1**	**287**	**226**	434	1144	0	296	713	227	1147	1667
	2	0	0	0	0	0	0								
	3	**223**	**388**	**0**	**109**	**426**	**1**								
	4	0	0	0	0	0	0								
	5	0	0	0	0	0	0								
	6	**0**	**229**	**0**	**186**	**0**	**0**								
3	1	0	0	0	0	0	0	1219	139	915	23	125	184	2259	346
	2	**840**	**139**	**564**	**23**	**109**	**143**								
	3	0	0	0	0	0	0								
	4	**158**	**0**	**140**	**0**	**16**	**41**								
	5	**221**	**0**	**211**	**0**	**0**	**0**								
	6	0	0	0	0	0	0								
4	1	0	0	0	0	0	0	297	0	35	0	257	22	589	22
	2	0	0	0	0	0	0								
	3	**297**	**0**	**35**	**0**	**257**	**22**								
	4	0	0	0	0	0	0								
	5	**0**	**0**	**0**	**0**	**0**	**0**								
	6	0	0	0	0	0	0								

Tab. 2.: Počty kapslí, původní strategie

zdrojový uzel	cílový uzel	datové kapsle - odeslání	datové kapsle - přeposlání	kapsle s aplikací - odeslání	kapsle s aplikací - přeposlání	žádosti o aplikaci - odeslání	žádosti o aplikaci - přeposlání	celkem datových kapslí - odeslání	celkem datových kapslí - přeposlání	celkem kapslí s aplikací - odeslání	celkem kapslí s aplikací - přeposlání	celkem žádostí o aplikaci - odeslání	celkem žádostí o aplikaci - přeposlání	celkem odesláno	celkem přeposláno
5	1	0	0	0	0	0	0	**427**	**368**	**5**	**117**	**537**	**198**	**969**	**683**
	2	0	0	0	0	0	0								
	3	**221**	**229**	**0**	**0**	**211**	**186**								
	4	**0**	**139**	**0**	**117**	**0**	**0**								
	5	0	0	0	0	0	0								
	6	**206**	**0**	**5**	**0**	**326**	**12**								
6	**1**	**225**	**206**	**0**	**0**	**250**	**212**	**454**	**551**	**62**	**329**	**453**	**264**	**969**	**1144**
	2	**0**	**0**	**0**	**0**	**0**	**0**								
	3	0	0	0	0	0	0								
	4	0	0	0	0	0	0								
	5	**229**	**345**	**62**	**329**	**203**	**52**								
	6	0	0	0	0	0	0								

Tab. 3.: Počty kapslí, původní strategie, pokračován tab. 2

zdrojový uzel	cílový uzel	datové kapsle - odeslání	datové kapsle - přeposlání	kapsle s aplikací - odeslání	kapsle s aplikací - přeposlání	žádosti o aplikaci - odeslání	žádosti o aplikaci - přeposlání	celkem datových kapslí - odeslání	celkem datových kapslí - přeposlání	celkem kapslí s aplikací - odeslání	celkem kapslí s aplikací - přeposlání	celkem žádosti o aplikaci - odeslání	celkem žádosti o aplikaci - přeposlání	celkem odesláno	celkem přeposláno
1	1	0	0	0	0	0	0	1142	0	525	0	1	0	1668	0
	2	**550**	**0**	**253**	**0**	1	**0**								
	3	0	0	0	0	0	0								
	4	0	0	0	0	0	0								
	5	0	0	0	0	0	0								
	6	**592**	**0**	**272**	**0**	**0**	**0**								
2	**1**	**202**	**493**	**0**	**1**	**203**	**50**	433	1064	144	125	461	125	1038	1314
	2	0	0	0	0	0	0								
	3	**231**	**348**	**57**	**50**	**258**	**75**								
	4	0	0	0	0	0	0								
	5	0	0	0	0	0	0								
	6	**0**	**223**	**87**	**74**	**0**	**0**								
3	1	0	0	0	0	0	0	1206	145	604	0	133	3	1943	148
	2	**802**	**145**	**330**	**0**	**107**	**3**								
	3	0	0	0	0	0	0								
	4	**144**	**0**	**117**	**0**	**26**	**0**								
	5	**260**	**0**	**157**	**0**	**0**	**0**								
	6	0	0	0	0	0	0								
4	1	0	0	0	0	0	0	289	0	14	0	231	12	534	12
	2	0	0	0	0	0	0								
	3	**289**	**0**	**14**	**0**	**117**	**12**								
	4	0	0	0	0	0	0								
	5	**0**	**0**	**0**	**0**	**114**	**0**								
	6	0	0	0	0	0	0								

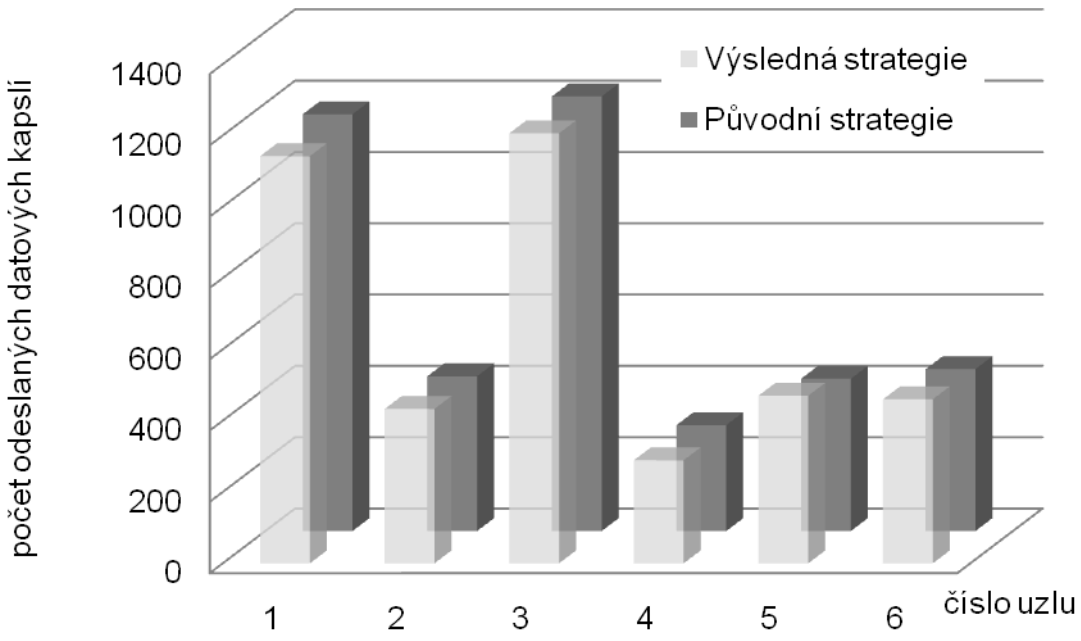
Tab. 4.: Počty kapslí, výsledná strategie

zdrojový uzel	cílový uzel	datové kapsle - odeslání	datové kapsle - přeposlání	kapsle s aplikací - odeslání	kapsle s aplikací - přeposlání	žádosti o aplikaci - odeslání	žádosti o aplikaci - přeposlání	celkem datových kapslí - odeslání	celkem datových kapslí - přeposlání	celkem kapslí s aplikací - odeslání	celkem kapslí s aplikací - přeposlání	celkem žádosti o aplikaci - odeslání	celkem žádosti o aplikaci - přeposlání	celkem odesláno	celkem přeposláno
5	1	0	0	0	0	0	0	**470**	**368**	**76**	**41**	**407**	**42**	**953**	**451**
	2	0	0	0	0	0	0								
	3	**260**	**223**	**0**	**0**	**157**	**0**								
	4	**0**	**145**	**73**	**41**	**0**	**0**								
	5	0	0	0	0	0	0								
	6	**210**	**0**	**3**	**0**	**250**	**42**								
6	**1**	**237**	**210**	**0**	**0**	**167**	**105**	**460**	**565**	**176**	**105**	**332**	**115**	**968**	**785**
	2	**0**	**0**	**0**	**0**	**161**	**0**								
	3	0	0	0	0	0	0								
	4	0	0	0	0	0	0								
	5	**223**	**355**	**176**	**105**	**4**	**10**								
	6	0	0	0	0	0	0								

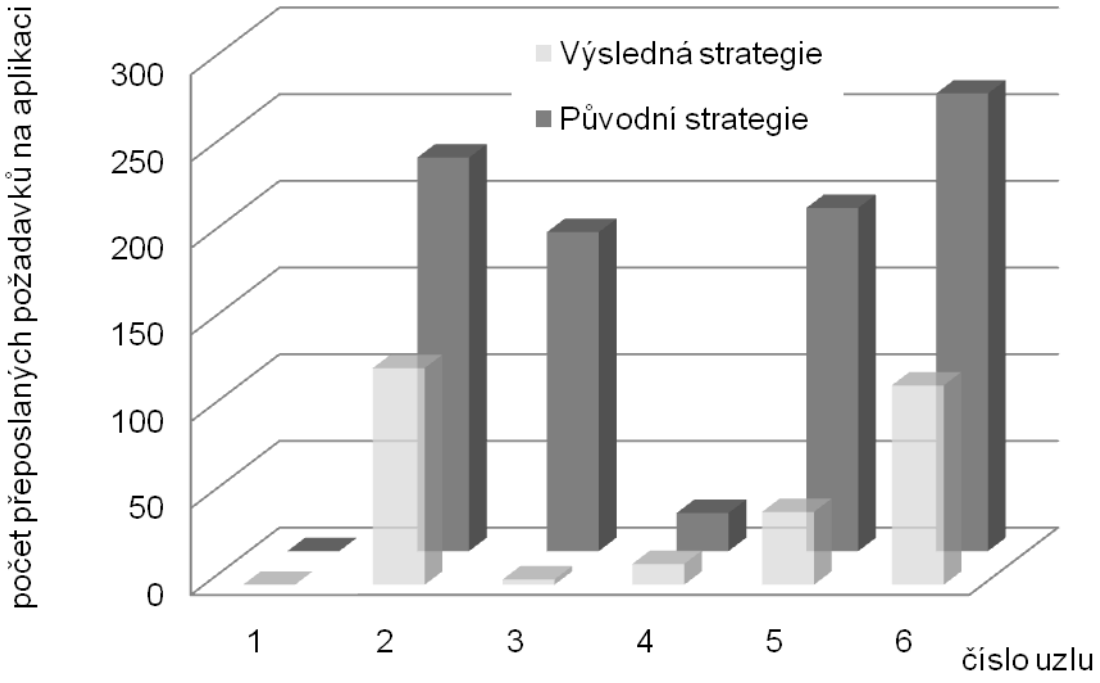
Tab. 5.: Počty kapslí, výsledná strategie, pokračování tab. 4

V předchozích tabulkách jsou zaznamenány počty všech typů kapslí odeslaných mezi jednotlivými uzly v průběhu testování. U každého uzlu je zvýrazněn řádek značící uzly, se kterými je spojen (pro ostatní uzly jsou pak počty kapslí přirozeně rovny nule). Následuje přehlednější zobrazení nejzajímavějších výsledků ve formě grafů.

Z grafu 1 je patrné, že v testech obou strategií uzly (a tedy i celou sítí) prošel podobný počet datových kapslí. Graf 2 pak názorně ilustruje pokles počtu žádostí o aplikace, které bylo nutné z jejich cílového uzlu poslat dál v případě nově vytvořené strategie distribuce kódu. Z toho vyplývá, že nová strategie přesněji určuje uzly, které budou o poskytnutí aplikace požádány a žádost pak není nutné přeposílat mezi mnoha dalšími uzly.



Graf 1.: Počet datových kapslí odeslaných jednotlivými uzly



Graf 2.: Počet žádostí o aplikaci, které uzly přeposlaly dál

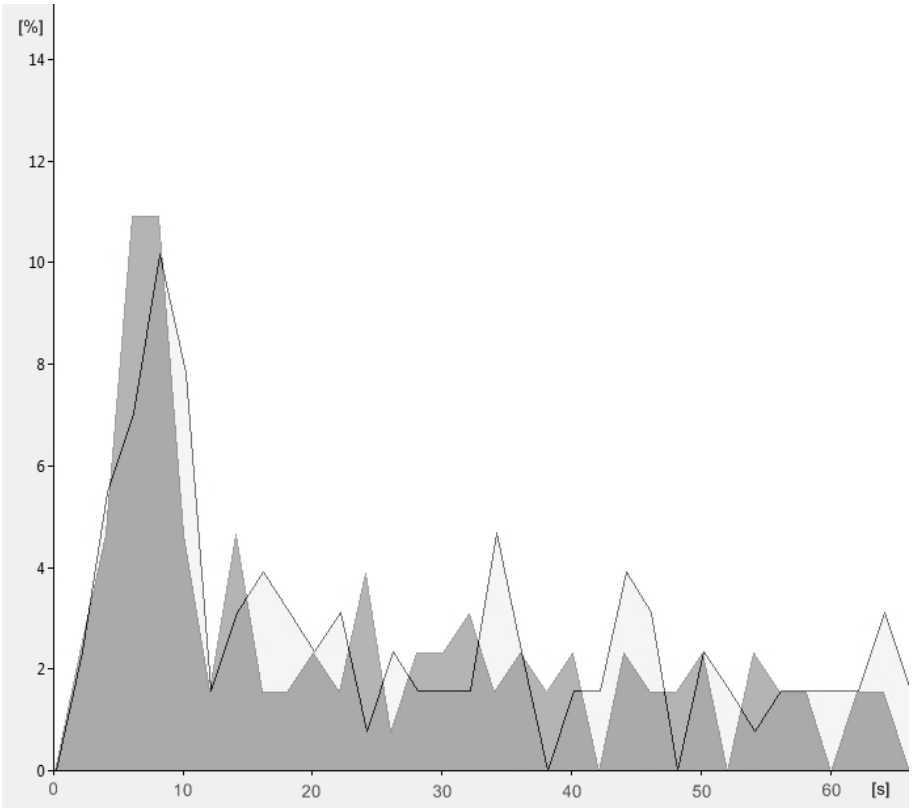
6.3. Průběh testování – zatížení uzlu

Dalším testovaným parametrem uzlů aktivní sítě SAN bylo zatížení uzlu. Testovací síť měla stejnou topologií jako v předchozím případě (viz obr. 7) a stejně tak se shodoval i způsob spouštění aplikací (viz kapitolu 6.2). Pro sledování byl použit nástroj „Reliability and Performance Monitor“, který je standardní součástí operačního systému Microsoft Windows (v tomto případě verze Windows XP). Protože každý uzel SAN běží ve formě samostatné Java Virtual Machine (JVM), byl u každého procesu JVM sledován parametr nazvaný „% Processor Time” (udává, jaké procento celkového času procesor strávil zpracováváním instrukcí daného procesu). Na každý uzel byl spuštěn právě jeden JVM.

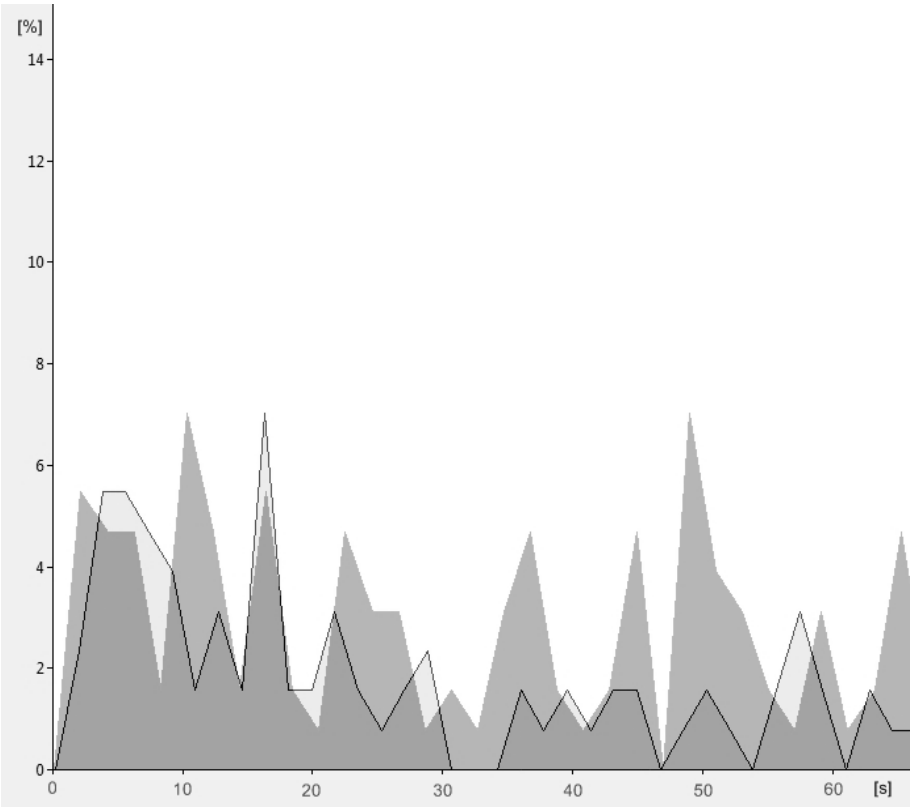
6.3.1. Výsledky testování – zatížení uzlu

Barevná podoba grafů 3 až 8 byla upravena s pomocí grafického editoru, hodnoty v grafech na obou osách zůstaly samozřejmě nezměněny. Na ose X je vynesen čas ve vteřinách, hodnota na ose Y je rovna velikosti změřené veličiny „% Processor Time”. Tmavě šedou barvou jsou zobrazeny výsledky změřené při použití původní strategie, světlejší barva značí hodnoty dosažené při použití výsledné strategie. Hodnoty jsou zobrazeny pouze pro první minutu běhu testu, frekvence vzorkování byla nastavena na 2 vteřiny.

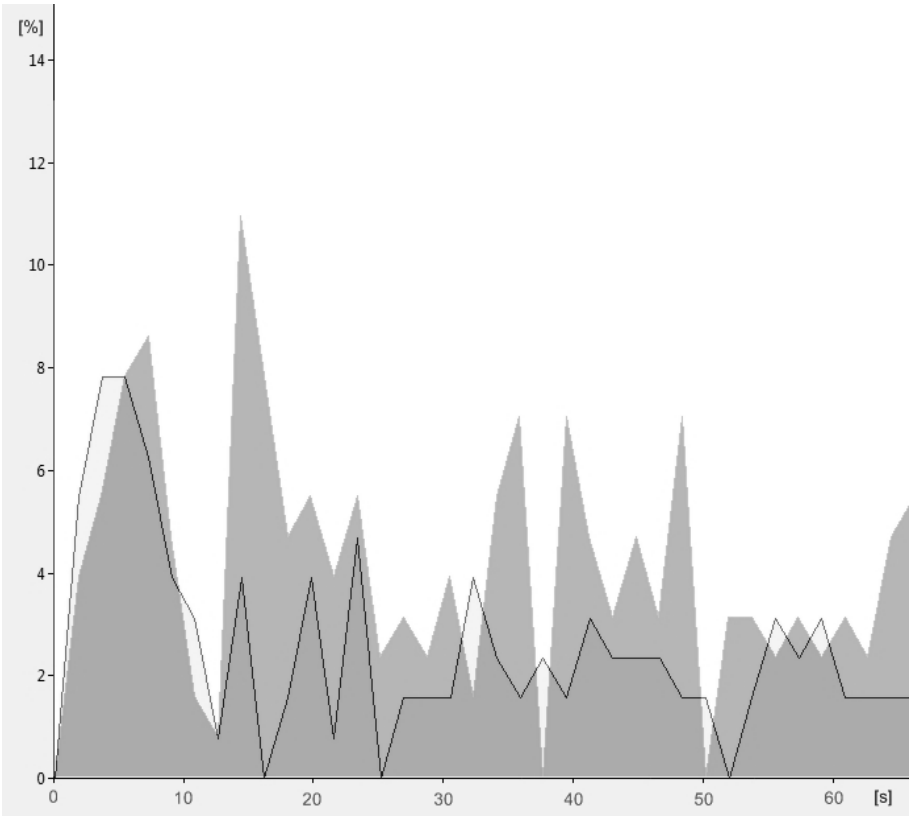
Z výsledků je vidět, že zatížení uzlů je při použití obou strategií v zásadě podobné, a implementace výsledné strategie nepřinesla v případě zatížení uzlů tak významný pokles jako v případě počtu přeposílaných kapslí s žádostmi o aplikace (viz kapitolu 6.2). Logicky byl očekáván pokles spotřeby procesorového času, jelikož byl menší počet kapslí ke zpracování. Možným vysvětlením mohou být nedostatky v implementaci plánovače. Zefektivnění plánovače je další naplánovaná úloha pro projekt SAN, po které by se dle teoretických očekávání měl projevit i pokles spotřeby procesorového času na distribuci aktivního kódu.



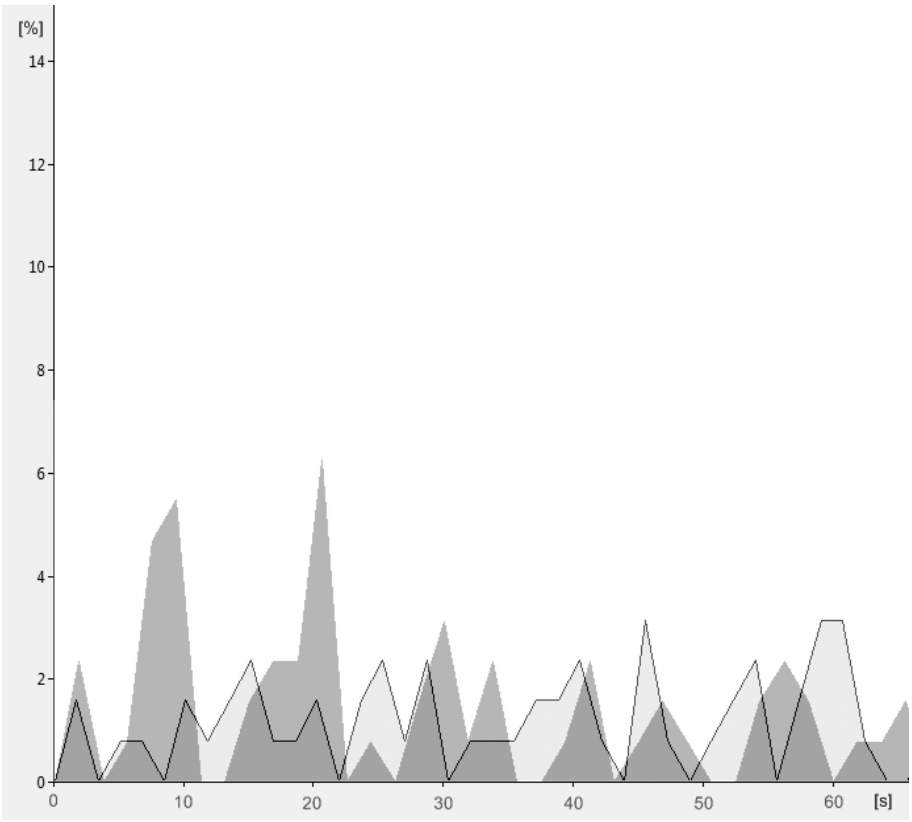
Graf 3.: Zatížení uzlu č. 1



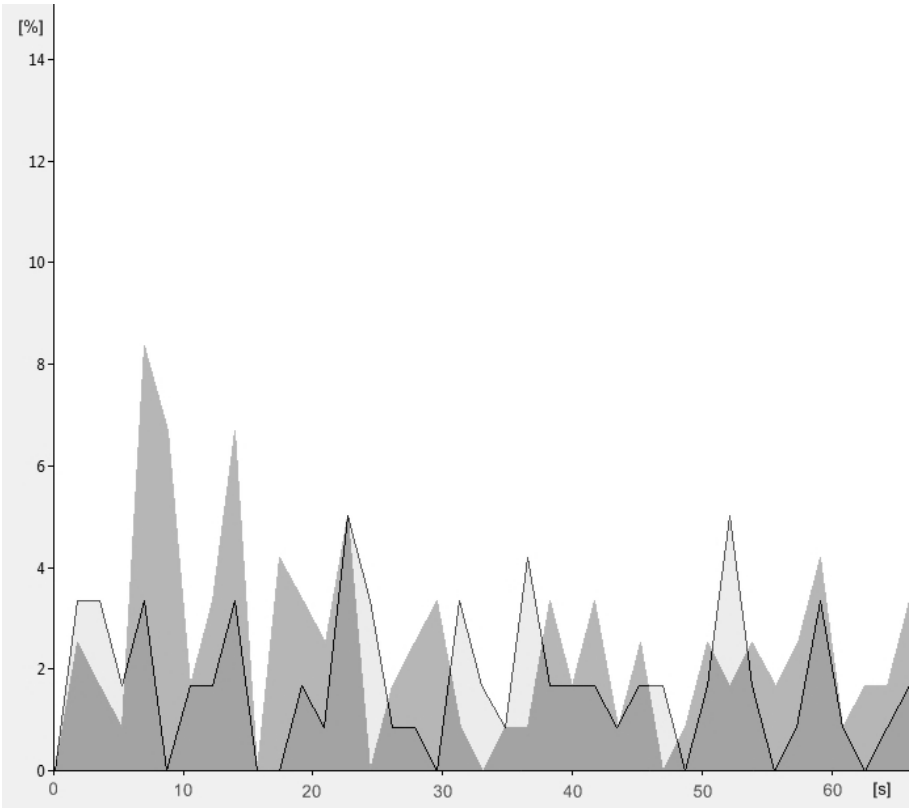
Graf 4.: Zatížení uzlu č. 2



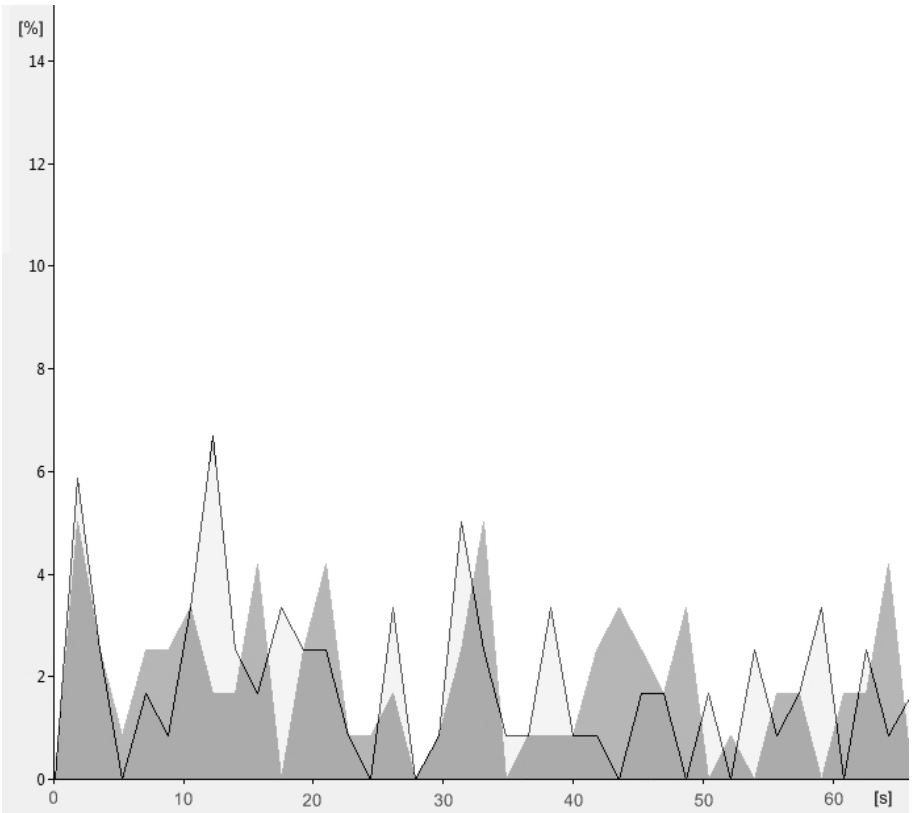
Graf 5.: Zatížení uzlu č. 3



Graf 6.: Zatížení uzlu č. 4



Graf 7.: Zatížení uzlu č. 5



Graf 8.: Zatížení uzlu č. 6

7. Závěr

Diplomová práce měla za cíl navrhnout a implementovat protokol distribuce kódu pro uzel aktivní sítě projektu SAN.

Po nezbytném úvodu do problematiky aktivních sítí obsahuje tento dokument detailní rozbor zadání práce a analytickou část, ve které je vybráno nejvhodnější řešení. Následuje teoretický popis možností implementace v rámci již existující architektury uzlu SAN. Díky tomu je tato práce vhodná i pro zájemce o základní seznámení s konceptem aktivních sítí a především s projektem SAN.

Poté je popsána implementace konkrétního řešení ve formě modulu pro uzel SAN. Výsledkem je funkční uzel aktivní sítě, jehož zdrojový kód je k dispozici na přiloženém CD. Práci uzavírá kapitola dokumentující testování vytvořeného protokolu distribuce kódu a prezentace dosažených výsledků.

Práce poukazuje na několik oblastí, ve kterých je nutné uzel SAN dále zdokonalovat. Především je to bezpečnost uzlu a plánování vykonávaného kódu.

V diplomové práci se podařilo splnit zadané cíle. Výsledkem je vylepšená důležitá funkčnosti projektu SAN, který díky tomu může sloužit k dalším plánovaným výzkumům a experimentům, stejně jako i výzkumům nově umožněným právě touto přidanou funkcionalitou, na poli aktivních sítí. V neposlední řadě se tím projekt SAN dostává opět o krok blíže k plně funkčnímu prototypu uzlu aktivní sítě.

Přehled zkratek

SAN	Smart Active Node
DARPA	Defense Advanced Research Projects Agency
JVM	Java Virtual Machine
XML	Extensible Markup Language
SPK	Smart Active Node Package

Literatura

[TSWM97] TENNENHOUSE, D., SMITH, J., SINCOSKIE, D., WETHERALL, D., MINDEN, G. *A Survey of Active Network Research,* IEEE Communications Magazine, 1997, vol. 35, str. 80-86.

[Rej08] REJDA, M. *Server aktivní sítě*, Plzeň, 2008, Diplomová práce na Fakultě aplikovaných věd Západočeské univerzity v Plzni na katedře informatiky a výpočetní techniky. Vedoucí diplomové práce Ing. Tomáš Koutný, Ph.D.

[WGT99] WETHERALL, D., GUTTAG, J., TENNENHOUSE, D. *ANTS: Network Services Without the Red Tape*. Computer, April 1999, vol. 32, no. 4, s. 42-48.

[WGT98] WETHERALL, D., GUTTAG, J., TENNENHOUSE, D. *ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols*, IEEE Open Architectures and Network Programming Conference, 1998. s. 117-129.

[Alex98a] ALEXANDER, D. *The SwitchWare Active Network Architecture*. IEEE Network Special Issue on Active and Controllable Networks, 1998, vol. 12, no. 3, s. 29-36.

[Alex98b] ALEXANDER, D. *The SwitchWare Active Network Implementation*. The 1998 ACM SIGPLAN Workshop on ML. Baltimore, Maryland, USA, 1998.

[MB99] MERUGU, S., BHATTACHARJEE, S., CHAE, Y., SANDERS, M., CALVERT, K., ZEGURA, E. *Bowman and CANEs: Implementation of an Active Network*. Invited paper at 37th Annual Allerton Conference, 1999.

[YDS96] YEMINI, Y., DA SILVA, S. *Towards Programmable Networks*. Proceedings of IFIP/IEEE International Workshop on Distributed System: Operations and Management, 1996.

Příloha A – Uživatelská příručka

Formát souboru s nastavením uzlu

Soubor s nastavením je textový soubor a data jsou v něm uložena ve formátu xml. Název souboru může být libovolný. Pokud je jiný než výchozí název `settings.xml` nebo je umístěn jinde než v kořenovém adresáři uzlu SAN, musí být jeho název, resp. i cesta k souboru, uvedena jako parametr při spuštění uzlu.

Ukázka minimálního funkčního souboru s nastavením:

```
<?xml version="1.0" encoding="UTF-8"?>
<node name="uzel1">
  <interface name="eth0">
    <identity>
      <node>
        00000000-0000-0000-0000-000000000001
      </node>
      <network>
        00000000-0000-0000-0000-000000000001
      </network>
    </identity>
    <broadcast ip="230.0.0.1" port="8000" />
    <method value="0" priority="1"
    <method value="1" priority="0" />
  </interface>

  <defaultGateways>
    <gateway network="00000000-0000-0000-0000-000000000001">
      <identity>
        <node>
           00000000-0000-0000-0000-000000000002
        </node>
        <network>
           00000000-0000-0000-0000-000000000001
        </network>
      </identity>
      <interface>
        <node>
          00000000-0000-0000-0000-000000000001
        </node>
        <network>
          00000000-0000-0000-0000-000000000001
        </network>
```

```
      </interface>
    </gateway>
  </defaultGateways>
  <logging output="CONSOLE" level="DEBUG" />
  <console type="text" />
  <codeRepository>
    codeRepository1
  </codeRepository>
</node>
```

Celé nastavení je obsažené v kořenovém elementu `node` s parametrem `name`. Vlastnosti každého síťového rozhraní tohoto uzlu jsou popsány v příslušném elementu `interface`. Jejich povinnou součástí je identifikace rozhraní pomocí elementu `identity` obsahujícím číslo rozhraní a číslo sítě. Nastavení uzlu dále obsahuje definici výchozí směrovací cesty v elementu `defaultGateways`. Tato cesta je poté vybírána na základě sítě, do které je nutné kapsli poslat, proto je nutnou součástí každé cesty číslo sítě v parametru `network` u elementu `gateway`. Element `identity` u výchozí cesty určuje cílové rozhraní, `interface` pak lokální rozhraní uzlu, které bude použito k odeslání kapsle. Obsah elementu `codeRepository` určuje cestu k adresáři, který uzlu slouží jako úložiště aplikací.

Možností rozšíření nastavení uzlu je přidání směrování. Jedná se o nepovinnou součást, ovšem zároveň o jedinou možnost, jak nyní definovat v aktivní síti uzlů SAN směrování (kromě povinné výchozí cesty z každého uzlu). Takto vypadá definice jedné cesty:

```
<routing>
  <route>
    <to>
      <identity>
        <node>
          00000000-0000-0000-0000-000000000009
        </node>
        <network>
          00000000-0000-0000-0000-000000000001
        </network>
      </identity>
    </to>
    <gateway>
```

```
      <identity>
        <node>
          00000000-0000-0000-0000-000000000003
        </node>
        <network>
          00000000-0000-0000-0000-000000000001
        </network>
      </identity>
    </gateway>
    <interface>
      <identity>
        <node>
          00000000-0000-0000-0000-000000000001
        </node>
        <network>
          00000000-0000-0000-0000-000000000001
        </network>
      </identity>
    </interface>
  </route>
</routing>
```

Element `to` určuje cílové síťové rozhraní, `gateway` definuje rozhraní, které v cestě bezprostředně následuje po tomto uzlu a element interface určuje lokální síťové rozhraní, které bude použito jako odchozí pro všechny kapsle odesílané na uzel s cílovým síťovým rozhraním.

Spuštění uzlu

Pro spuštění lze použít předpřipravené skripty `run[celé_číslo].bat` v adresáři s projektem. Tomuto skriptu odpovídá příslušný soubor s nastavením `settings[stejné_celé_číslo].xml`.

Webová stránka se statistikou

Jak bylo popsáno v kapitole 5.3.2 (podkapitola Zobrazení statistiky kapslí a aplikací), pokud je název souboru s nastavením ve tvaru `settings[celé_číslo].xml`, pak uzel spustí na adrese `http://localhost` a portu číslo `5560 + [celé_číslo_z_názvu_souboru]` webový server. Jedinou stránkou přístupnou na tomto serveru je stránka se statistikami provozu na síťových rozhraních uzlu a stavu různých částí uzlu. Konkrétně se jedná o tyto informace:

- Statistika počtu kapslí odeslaných všemi rozhraními uzlu. Počty jsou rozděleny podle typu kapsle (datová, přenášející aplikaci a kapsle s požadavkem o aplikaci) a podle toho, zda se jedná o kapsli nově vytvořenou uzlem nebo přeposlanou kapsli vzniklou na jiném uzlu.
- Stejné informace o počtu kapslí jako v předchozím bodě, ale rozdělené podle síťových rozhraní, na které byly kapsle odeslány.
- Obsah směrovací tabulky uzlu. Přehled uložených cest ke konkrétním síťovým rozhraním i výchozích cest pro jednotlivé sítě.
- Přehled identifikátorů síťových rozhraní sousedních uzlů přímo spojených s rozhraními tohoto uzlu
- Informace o aplikacích. Ke každé aplikaci, která je na uzlu uložena nebo uzlem prošla alespoň kapsle určená pro tuto kapsli je zobrazeno jméno aplikace, identifikátor aplikace (hash), zda je aplikace v současné době uložena na uzlu a identifikátory síťových rozhraní, ze kterých přišla kapsle s aplikací nebo daty pro aplikaci a na které byla tato aplikace nebo její data poslána (viz také popis modulu Traffic Logger, kapitola 4.4.2).