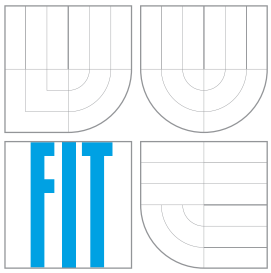
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

CLOUDOVÝ SERVER PRO SPRÁVU ROBOTŮ

ROBOT MANAGEMENT CLOUD SERVER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATOUŠ JEZERSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK MATERNA

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Jezerský Matouš**
Obor: Informační technologie
Téma: **Cloudový server pro správu robotů**
Cloud-Based Server for Robot Management
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s principy návrhu a implementace cloudových služeb.
2. Seznamte se s robotickým operačním systémem (ROS) a existujícími nástroji pro správu robotů.
3. Proveďte analýzu a návrh serverové aplikace.
4. Realizujte navržené řešení.
5. Ověřte funkčnost řešení na serveru a robotech skupiny Robo@FIT.
6. Porovnejte a zhodnoťte dosažené výsledky. Navrhněte vhodné způsoby využití vašeho řešení.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Materna Zdeněk, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem tohoto projektu je zefektivnit a zabezpečit uživatelský přístup k robotům využívajícím platformu ROS a serverům, které na nich běží, bez nutnosti rekonfigurace při změně sítě či potřeby přímého VPN připojení, a také s důrazem na co nejmenší softwarové požadavky na uživatele a roboty. Výsledkem je open-source aplikace, která toto umožňuje. Řešením je vytvoření reverzního proxy serveru mezi klientskými aplikacemi a servery běžícími na robotech (dále Dispatcher), pro propojení klientů a robotů je využito dvou OpenVPN sítí a jako bezpečné rozhraní s autentizací uživatelů je využito existujícího systému RMS, který je rozšířen o rozhraní pro Dispatcher. Dispatcher mimo jiné také sbírá informace o všech připojených robotech pro snadné zobrazení. Tato sada programů tedy umožnuje oprávněnému uživateli připojení k robotům v libovolných sítích bez potřeby klientského softwaru mimo klienta OpenVPN a webového prohlížeče.

Abstract

The goal of this project is to make robots running the ROS platform, and the servers running on them, more accessible to the users, without the need of reconfiguration after a change of network or the need of direct VPN connection, as well as with the emphasis on minimal software requirements for both users and robots. The result is an open-source application, which provides that. The solution is a reverse proxy server, put inbetween the clients and the servers running on robots - Dispatcher. For connection between clients and robots, two OpenVPN networks are used, and as a secure user interface, the existing system RMS is used and extended with Dispatcher interface. Among other things, Dispatcher collects information about all connected robots so a summary can be displayed. This set of software tools therefore allows an authorized user to connect to robots in various networks, without the need of other client software apart of OpenVPN client and web browser.

Klíčová slova

správa robotů, reverzní proxy, Dispatcher, cloud, robotika, počítačové sítě

Keywords

robot management, reverse proxy, Dispatcher, cloud, robotics, networking

Citace

JEZERSKÝ, Matouš. *Cloudový server pro správu robotů.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Materna Zdeněk.

Cloudový server pro správu robotů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Materny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Matouš Jezerský
17. května 2016

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce, Ing. Zdeňku Maternovi, za jeho rady a pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Účelem tohoto projektu je umožnit spojení uživatelů a robotů skrze jeden server, a to nezávisle na tom, v jakých sítích se uživatelé či roboti nacházejí, s důrazem na bezpečnost jejich komunikace. Nabízené řešení může být využito při výuce nebo práci s více roboty v laboratoři nebo i ke vzdálené práci s roboty, což by například studentům umožňilo pracovat s roboty vzdáleně a poskytovalo nejen přístup a řízení robotů, ale také možnost tvorby nových aplikací, pro které by toto řešení vytvářelo síťovou infrastrukturu a poskytovalo vrstvu zabezpečení, nezávisle na tom, jaký aplikační protokol je využit. Za robota je v této práci považováno zařízení s platformou ROS [13].

Navrhovaná varianta řešení se skládá ze tří základních částí – sítě s využitím VPN (2.3), Dispatcheru a RMS [1] (2.2). VPN propojuje roboty a server, je tedy možné využít této existující sítě při vývoji nových aplikací, kdy složitější výpočty mohou probíhat na serveru, bez nutnosti nastavování NAT (2.5), sítě firewall, apod. Dispatcher slouží ke směrování síťového provozu dle aktuální konfigurace, která je dynamicky měněna pomocí rozšíření RMS, které v existujícím systému implementuje rozhraní pro lokální komunikaci mezi PHP interpretem a Dispatcherem, a také umožňuje graficky zobrazovat data přijímaná z Dispatcher serveru.

Jelikož na robotech běží různé serverové aplikace, jako např. *roscore*, *rosbridge*, *mjpeg* a podobné [9], nelze se na ně jednoduše připojit, pokud jsou za bránou firewall, nebo je v síti využita NAT. K řešení tohoto problému by v běžném případě stačila síť VPN, spojující uživatele a roboty, ovšem přímé spojení uživatele a robota využívajícího platformu ROS přináší bezpečnostní rizika, protože *roscore* umožňuje neautorizovaný přístup, to by tedy znamenalo, že každý uživatel by měl neomezený přístup k robotům, ke kterým by byl připojen. Zde tedy Dispatcher slouží jako forma zabezpečení, případně by jej také bylo mozné rozšířit o řízení různých úrovní přistupu (read-only), chtěli bychom zasahovat i do jiných protokolů, než na které mají vliv rozhraní v RMS.

Pokud ovšem vložíme Dispatcher mezi uživatele a roboty, vzniká problém se způsobem přesměrovávání komunikace. Dispatcher přesměrovává síťový provoz podle aktuální konfigurace, kdy je zvolen pár IP adres – zdrojová (klient) a cílová (robot). Veškerá komunikace iniciovaná klientem je tedy dle jeho IP adresy přesměrována na adresu robota, dle tohoto páru adres. Pokud je ovšem více klientů v síti se sdílenou adresou, dochází k problému, a je nutné vyřešit způsob identifikace klientů. Jednou z možností by byla úprava klientských aplikací, což by ale znamenalo nutnost změny každé aplikace, která by byla použita při komunikaci s robotem. Zde se opět nabízí využití sítě VPN, která klientům zajistí unikátní adresu a také další vrstvu zabezpečení.

V ROS open-source komunitě (wiki.ros.org) zatím neexistuje veřejně dostupný software

či softwarový balík, který by toto jednoduše umožňoval a zároveň poskytoval dostatečné zabezpečení. Jednou z alternativ tohoto řešení je využití softwaru *robots in concert* a systému RMS, toto ovšem vyžaduje úpravu klientského softwaru a je aplikovatelné pouze na komunikaci v systému ROS, tedy topic subscription/publishing. Dále je možné využít dříve zmiňovanou síť VPN, ta ovšem nezajišťuje zabezpečení *roscore*, bylo by tedy nutné vytvořit firewall pravidla pro omezení přístupu k *roscore* a následně spouštět webserver či autentizační aplikaci na každém robotovi zvlášť. Což by mohl být ovšem problém, vzhledem k tomu, že komunikace mezi prvky ROS probíhá částečně na náhodných portech. Autentizaci by bylo možné centralizovat např. s využitím LDAP serveru. To ovšem stále neřeší způsob komunikace uživatele s robotem, jelikož by nejspíš bylo nutné upravit klientské programy pro odesílání klíče či identifikátoru pro autentizaci uživatele.

I přes snahu co nejméně zatěžovat klienta softwarovými požadavky, je nutné, kromě webového prohlížeče nebo klientské aplikace třetí strany, nainstalovat klienta OpenVPN.

Kapitola 2

Související a použité technologie

2.1 ROS

Projekt je cílen na roboty, na kterých je využita platforma **Robotic Operating System (ROS)**. Jedná se o open-source framework pro zjednodušení tvorby programů a aplikací zaměřených na robotiku. ROS je sada nástrojů, knihoven a konvencí, standardně využívaná na operačním systému Ubuntu a Mac OS X. Samotný framework je jazykově nezávislý, je možné jej implementovat v různých moderních jazycích. Známe implementace jsou např. v jazycích Python, C++ a Lisp.

Dle popisu na webových stránkách projektu ROS [7] se jedná o meta-operační systém pro roboty, poskytující služby jako abstrakci hardwaru, nízkoúrovňové ovládání zařízení, implementace běžných standardně využívaných nástrojů v robotice, zasílání zpráv mezi procesy a správu softwarových balíků. Dále také poskytuje nástroje a knihovny pro získávání, sestavování, psaní a spouštění kódu na několika zařízeních. ROS implementuje síť peer-to-peer procesů, které také mohou být rozděleny mezi několik zařízení, které lze řídit přes síťový protokol tohoto systému.

ROS je navržen minimalisticky, kód psaný pro tento framework nezasahuje do zásadních funkcí programu (jako např. main) a je jej možné využívat i spolu s jinými frameworky pro robotiku, jako jsou např. *OpenRAVE*, *Orocos* nebo *Player*. Zároveň také poskytuje framework pro tvorbu testů *rostest*.

Jednou ze zásadních částí ROS je proces *roscore*, který, mimo jiné, zajišťuje komunikaci mezi jednotlivými procesy.ROS umožňuje komunikaci pomocí systému zpráv, který dělí zprávy do skupin (*Topics*) a také rozlišuje komponenty, které zprávy vysílají (*Publishers*) a které zprávy přijímají (*Subscribers*).

V systému ROS funguje *roscore* jako nástroj pro zprostředkování komunikace mezi jednotlivými prvky – *ROS Nodes*. Chceme-li přijímat nějaká data zprostředkovaná *roscore*, je třeba se přihlásit na daný *Topic*. Možností jak toto provést je několik, v závislosti na implementaci frameworku ROS. Chceme-li získávat data například ze skriptu psaném v jazyce Python, je k dispozici modul *rospy*, který poskytuje nástroje na vytvoření *Subscriber* procesu. Parametry, které tento proces vyžaduje, jsou adresa *roscore*, v jehož síti jsou zprávy pro daný *Topic* publikovány, a také samotnou identifikaci daného *Topic*. Poté dojde k připojení na *roscore* a modul začne získávat data. Data jsou přijímána vždy po tom, co se změní, tedy vždy, kdy nějaký proces zašle na zvolený *Topic* zprávu.

Chceme-li naopak zprávu vysílat, tedy vytvořit proces *Publisher*, je postup podobný. Nejprve se připojíme na *Topic*, na který chceme zprávu vysílat a obsah zprávy, kterou chceme odeslat.

Vytváříme-li nebo přijímáme zprávu, je třeba znát typ této zprávy. Typů zpráv v ROS je mnoho a nebylo by relevantní se zde všemi zabývat, nicméně jeden z příkladů takovéto zprávy může být např. objekt `std_msgs/String`, který obsahuje pouhý řetězec, nebo také objekt `geometry_msgs/Twist`, který se využívá při řízení pohybu různých součástí robota apod. Objekt `geometry_msgs/Twist` by mohl vypadat například následovně:

linear:
  x: 1.0
  y: 0.0
  z: -0.5
angular:
  x: 0.0
  y: 0.0
  z: 0.0

2.2 RMS

V tomto projektu je využito také systému **Robot Management System (RMS)** [1], který poskytuje nástroje k tvorbě a používání webových rozhraní pro řízení robotů, přijímání obrazových dat, apod. Tento systém zajišťuje autentizaci uživatelů a správu rozhraní k ovládání robotů, a také poskytuje rezervační systém, kde je možné mezi uživateli rozvrhnout čas pro práci s roboty. Ovládací rozhraní umožňuje řízení robotů se systémem ROS pomocí javascript knihovny *roslibjs* [3], která ke komunikaci využívá JSON objekty a *rosbridge* server [2] na straně robota. *Rosbridge* převádí JSON objekty na ROS zprávy a naopak.

2.2.1 Model-View-Controller (MVC)

RMS využívá návrhového vzoru MVC, který je popisován také jako softwarová architektura, jelikož má na aplikace větší vliv než běžné návrhové vzory – týká se totiž celkového principu implementace, ne pouze jedné součásti. MVC byl původně navržen pro jakyk Smalltalk, ovšem časem se rozvinul dále. V dnešní době je již používán nezávisle na programovacím jazyce, především v oblasti tvorby uživatelských rozhraní. MVC rozděluje aplikaci do tří základních částí, které jsou odlišeny tak, aby zásah do jedné z nich vyžadoval minimální úpravy v ostatních. Každá z těchto částí má v aplikaci specifický účel. Následující popis MVC je velmi stručný a zúžený dle jeho aplikace v RMS.

Model reprezentuje logiku, tedy různé operace, jako např. SQL dotazy nebo funkce, třídy či metody v PHP. Dále jej lze využít pro uchovávání stavu aplikace nebo k odesílání událostí do *View*.

View zpracovává výstup, poskytuje uživateli viditelné rozhraní. View obsahuje především HTML kód, případně volá *Controller* pro změnu stavu, nebo očekává stavové události od *Model*.

Controller zpracovává vstup. Může požádat o zobrazení různých view, získat data z *Model*, nebo také zpracovávat uživatelský vstup (např. vyplnění formuláře apod.).

2.2.2 Základní prvky RMS

- *Environment* – kombinace adresy rosbridge serveru a MJPEG serveru, obsahuje tedy informace o IP adrese robota

- *Interface* – rozhraní pro řízení robota, např. robot s kamerou může mít rozhraní využívající MJPEG server pro příjem obrazu a řídit robota odesíláním dat do rosbridge pomocí ROS topic `/cmd-vel`. Jeden typ rozhraní se může vázat na několik *Environments*. Při vytvoření nového *Interface* v uživatelském rozhraní RMS je třeba vytvořit potřebné soubory, tedy napsat kód vlastního rozhraní. Tyto soubory jsou (vycházíme z kořenového adresáře RMS): `app/Controller/nazevInterfaceController.php` a veškeré *View* v `app/View/nazevInterface`, kde „nazev“ je název daného *Interface*.

- *Study* – informace o rezervaci. *Study* obsahuje informace o délce možné rezervace (např. 1 hodina), rozsahu platnosti, tedy v jakém časovém rozmezí lze dobu zarezervovat (např. kdykoliv v únoru, jsou-li ostatní podminky splněny). Dále také parametry *On-The-Fly*, který při zvolení umožňuje opětovné připojení do již probíhající rezervované instance, a dále parametry určující, zda se jedná o jednorázovou studii nebo lze rezervaci opakovat, parametr umožňující anonymní přístup a také jeden, který určuje, zda může probíhat více instancí zároveň (jsou-li splněny ostatní podmínky).

- *Condition* – spojuje a obsahuje informace o přiřazených *Study*, *Interface*, *Environment* a informaci o tom, kolik je rezervací je možno vytvořit (*Slots*). Právě zde je určeno, jaké rozhraní je pro řízení robota aplikováno. Je tedy možné rozlišit rezervace pro řízení robota a třeba pouze přijímání obrazu tak, že jeden uživatel robota řídí a ostatní pouze sledují obrazová data.

- *Slots* – časové bloky, po které je možné provést danou rezervaci *Slots* se váží na *Study* a *Condition*.

2.2.3 Systém rezervací

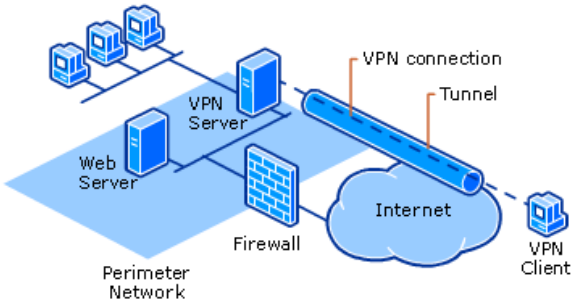
Pokud chce uživatel pracovat s robotem, prvním krokem je rezervace. Přehled možných rezervací je zobrazen na stránce `MY ACCOUNT`, která je zároveň výchozí stránkou po přihlášení, v sekci `AVAILABLE USER STUDIES`. Možné rezervace mohou vytvářet a měnit pouze uživatelé patřící do administrátorské skupiny. Běžný uživatel má tedy k dispozici pouze seznam rezervací, které on sám může využít. Po zarezervování práce s robotem (jsou-li splněny všechny podmínky), dojde k vytvoření záznamu o rezervaci a uživatel je přesměrován na rozhraní, které se na tuto rezervaci váže.

Pokud uživatel rozhraní pro interakci s robotem opustí, je možné pokračovat v existující práci s robotem, umožňuje-li to daný typ rezervace, pomocí odkazu, který je umístěn v sekci `SCHEDULED USER STUDIES` na stránce `MY ACCOUNT`.

2.3 VPN

2.3.1 Technologie VPN

Technologie VPN a její součást, tunelování, jsou velmi dobře popsány na stránkách informačního zdroje Microsoft TechNet [8].



Obrázek 2.1: Ilustrace VPN připojení a tunelu (převzato z Microsoft TechNet [4]).

Virtuální síť

Virtuální privátní síť (Virtual Private Network) popisuje jako rozšíření privátní sítě, která zahrnuje spojení napříč sdílenými či veřejnými sítěmi, jako třeba internet. VPN umožňuje odesílání dat mezi dvěma počítači napříč sdílenou nebo veřejnou sítí takovým způsobem, který emuluje vlastnosti přímého (point-to-point) privátního spojení.
K emulaci přímého spojení VPN zapouzdřuje nebo zaobaluje přenášená data přidáním hlavičky, která obsahuje informace o směrování, což datům umožňuje cestovat skrze sdílenou nebo veřejnou síť tak, aby tato data mohla dosáhnout své destinace. Aby bylo možné emulovat privátní spojení, odesílaná data jsou šifrována. Pakety které jsou zachyceny během cesty mezi zdrojem a cílem jsou tedy nedešifrovatelné bez šifrovacích klíčů, které VPN využívá. Ta část připojení, v níž jsou data zapouzdřena se nazývá tunel, a ta část připojení, kde jsou data šifrována, se nazývá VPN spojení.

Tunelování

Tunelování popisuje jako metodu využití síťové infrastruktury k přenosu dat, která jsou určena pro jistou síť, přes síť jinou. Data, která jsou určena k přenosu mohou být rámce nebo pakety různých protokolů. Namísto odesílání rámce nebo paketu tak, jak je vytvořen ve zdrojovém síťovém uzlu, jej tunelovací protokol zapouzdří v rámci nebo paketu s pomocí hlavičky navíc. Tato dodatečná hlavička poskytuje směrovací informace umožňující zapouzdřeným datům dosáhnout svého cíle skrze síť, která je prostředníkem, tedy síť, skrze kterou připojení tunelujeme.
Zapouzdřená data jsou poté směrována mezi koncovými prvky tohoto spojení. Tunelem nazýváme tedy logickou cestu, skrze kterou se data přenášejí a směrují skrze síť, přes kterou data tunelujeme.

Protokoly

Tunelovací protokoly popisuje v závislosti na typu vrstvy, přes kterou tunelujeme. Tunelovací technologie mohou být založeny buďto na 2. nebo 3. vrstvě. Tyto vrstvy odpovídají

referenčnímu modelu OSI. Tunelovací protokoly na druhé vrstvě OSI modelu odpovídají datové vrstvě a využívají rámce jako jejich datovou jednotku. Mezi takové protokoly patří např. PPTP (Point-to-Point Tunneling Protocol) a L2TP (Layer 2 Tunneling Protocol), které zapouzdřují data do PPP (Point-to-Point Protocol) rámců určených k odeslání přes tunelovanou síť. Protokoly 3. vrstvy odpovídají vrstvě síťové, a pracují tedy s pakety. Jedním z takovýchto protokolů je např. tunelovací režim IPSec, který zapouzdřuje IP pakety dodatečnou IP hlavičkou před tím, než je odešle přes tunelovanou síť.
Open-source VPN implementace OpenVPN ovšem nevyužívá žádný z těchto protokolů, nýbrž svůj vlastní - OpenVPN Protocol.

2.3.2 Využití v projektu

V projektu je využito sítí VPN (Virtual Private Network), které propojují roboty a server, a také uživatele a server. Tyto sítě souží ze strany robotů především k překonání NAT a bran firewall, jelikož na robotech běží různé aplikační servery a přímé připojení na ně by nastavení těchto síťových prvků vyžadovalo. Ze strany uživatelů jsou využity jako vrstva bezpečnosti a také pro jednoznačnou identifikaci klientů, nezávisle na využitém aplikačním protokolu, který je přes tuto síť přenášen. Na straně serveru pak lze jednoznačně určit, ze kterého zařízení požadavek přišel, bez nutnosti modifikace aplikačního protokolu, což umožňuje snadné využití aplikací třetích stran bez jakýchkoliv úprav.

Síť VPN vytváří mezi serverem a klientem (uživatelem nebo robotem) připojení, které mezi nimi umožňuje, s využitím tunelování, přenést libovolnou síťovou komunikaci. Připojení mezi klientem a serverem se emuluje tak, že jsou vytvořeny síťové adaptéry pro tunelování (např. v linuxu `tun/tap` adaptér), a přes tyto prvky je síťový provoz směrován. Tyto síťové adaptéry jsou v režii VPN aplikace, která veškerou komunikaci zapouzdřuje přidáním vlastní hlavičky, která obsahuje informace potřebné k identifikaci zdroje komunikace. Hlavička se liší dle implementace VPN protokolu.

Pro tento projekt se naskytlo využití několika VPN implementací, preferoval jsem open-source řešení, tím pádem se nabízí především OpenVPN a SoftEther VPN. SoftEther VPN umožňuje využití různých VPN protokolů, včetně OpenVPN protokolu, ovšem pro tento projekt je protokol OpenVPN dostačující, ostatní součásti tedy lze považovat za přebytečné. Jelikož tedy tyto další součásti nepotřebujeme, zvolil jsem OpenVPN.

2.4 Reverzní proxy server

Jelikož součástí návrhu je separace ROS od sítě uživatelů z bezpečnostních důvodů, je nutné tyto sítě nějakým bezpečným způsobem propojit. Možností by se naskytovalo několik, ovšem zde zmiňuji pouze tuto část, jelikož jsem ji dále zvolil pro implementaci.

Proxy server slouží jako prostředník mezi uživatelem a serverem, zprostředkovává komunikaci mezi těmito zařízeními a z pohledu serveru vystupuje jako klient [11]. Reverzní proxy server [16] funguje právě naopak. Předkládá tedy různé servery jako jeden. Reverzní proxy servery se používají k různým účelům, nejčastěji ovšem pro tzv. load balancing, indexování apod., tedy přistoupí-li uživatel na jisté místo v síti, tento proxy server vybere jeden z aplikačních serverů dle zátěže a prezentuje jej pod svou adresou. Problémem existujících open-source řešení ovšem je, že pracují především se statickou konfigurací, kterou je samozřejmě možné měnit, ovšem vyžaduje restart programu nebo znovunačtení konfiguračních souborů. Je proto třeba vytvořit takový, který bude umožňovat dynamickou rekonfiguraci, tedy změnu směrovacích pravidel za běhu.

Takovýto proxy server potřebuje nějaký zdroj informací o tom, jak má komunikaci mezi zařízeními směrovat, tedy jakousi směrovací tabulku. Standardně je tato tabulka definována právě konfiguračními soubory, nicméně dynamická rekonfigurace vyžaduje jinou implementaci, která je zmíněna v sekci 4.0.1.

2.5 Network Address Translation (NAT)

Dle RFC2663 [15] je NAT popsán jako metoda, podle které jsou IP adresy mapovány z jedné oblasti do druhé, ve snaze poskytnout transparentní směrování k hostitelům. Standardně jsou NAT zařízení používána k připojení izolované adresové oblasti s privátní neregistrovanou adresou k externí oblasti s globálně unikátní registrovanou adresou. Překlad adres umožňuje hostitelům v privátní síti transparentně komunikovat s cíly v externí síti a naopak. Toho dosahuje tak, že za běhu modifikuje adresu cílového uzlu a udržuje stav těchto modifikací tak, aby datagramy vztahující se k jednomu spojení (session), byly přesměrovány do správného cílového uzlu v jedné z oblastí.

Problémy způsobené tímto šetřením adres se v tomto projektu projevily v případě, že se chceme připojit na robota přes nějakou veřejnou adresu (třeba i přes dynamickou DNS), a bylo nutné je vyřešit. Právě takovým problémem je, že NAT výrazně ztěžuje způsob připojení k robotům v různých sítích, které tuto technologii využívají.

2.6 Cloud computing

Definicí cloud computingu je mnoho, bohužel neexistuje nějaký standard, který by jasně určoval, co přesne cloud computing je. Co je v této situaci dle mého názoru vhodné, je řídit se definicí poskytnutou těmi, kteří se cloud computingu skutečně věnují a ideálně právě těmi, kteří cloudové služby poskytují. Jedním z největších poskytovatelů cloudových řešení je Amazon se svým cloudem Amazon AWS (Amazon Web Services) - ti definují cloud computing jako dodání IT zdrojů a aplikací na požádání, a to skrze internet, kde uživatel platí pouze za tu část, kterou skutečně využívá [5].

Cloud dle G. Reese [14] je místo, kde lze využívat různých technologií ve chvíli, kdy jsou potřeba a přesně po tu dobu, jakou potřebujeme. Jedním z hlavních principů je právě to, že uživatel nemusí řešit nic mimo to, k čemu chce službu využívat. Stejně tak není třeba platit za něco navíc, co nebude využito. Cloud tedy tímto způsoben nabízí dynamickou škálovatelnost. Příkladem by mohlo být vytvoření serverové aplikace, která bude využívat minimální hardwarové prostředky z důvodů malého množství uživatelů, ovšem v budoucnu budeme očekávat, že se uživatelská báze rozšíří, není nutné předem platit za nevyužitý výkon a hardware, ale pouze za to, co momentálně využíváme, a to s tím, že ve chvíli, kdy počet uživatelů naroste, bude stačit pouze připlatit za lepší hardwarové prostředky.

Jinými slovy by se dalo říct, že pokud chceme vytvořit cloudový server, nemusíme rovnou investovat do fyzického hardwaru, který samozřejmě lze s časem rozšířit a aktualizovat, nicméně při pozdějším poklesu potřeby výkonu jej nemůžeme jednoduše prodat zpět, ale můžeme si pouze pronajmout virtuální hardware, jehož parametry můžeme dle potřeby měnit. Toto je samozřejmě pouhý příklad, cloud computing neznamená pouze virtuální hardware, což je zmíněno dále.

Cloud může být jak software, tak infrastruktura. Může to být aplikace, ke které přistupujeme skrze web nebo server, který je využíván pouze v případě, je-li potřeba. Nebo třeba celý virtuální počítač, běžící na škálovatelném virtuálním hardwaru - tedy na fyzickém

hardwaru, kde právě tento počítač má vymezeny povolené prostředky, které lze dynamicky měnit.

Pro tento projekt je cloudovým řešením právě umístění serveru, ke kterému se roboti a uživatelé připojují, na libovolném stroji, tedy např. právě do cloudové infrastruktury. Umístíme-li server právě tam, nabízí se široké možnosti jeho využití. Nejen, že lze přizpůsobit potřebné hardwarové nároky s tím, kolik robotů a uživatelů jej bude využívat, ale také můžeme jednoduše tento server škálovat v případě, že bychom na něj chtěli umístit libovolné další aplikace, a to kupříkladu takové, které by mohli roboti využívat k ulehčení zátěže na nich samotných. Tímto je myšleno, že pokud robot např. zpracovává obraz, stačilo by, aby na robotovi docházelo ke snímání obrazu a jeho kompresi, a tento obraz by byl dále zpracován až na straně serveru, který by měl k dispozici více hardwarových prostředků.

2.6.1 Typy cloud computingu

SaaS

Software as a service (SaaS) je v podstatě termín, kterým je popisován software v cloudu, i když všechny SaaS systémy nemusí být nutně cloudové, většina z nich je. Jedná se o způsob nasazení softwaru především formou webových technologií. Čímž tento systém zpřístupňuje i komplikovanější software pomocí webového prohlížeče, který je dostupný na velmi široké škále zařízení. Uživatele SaaS tedy nemusí zajímat, na jakém operačním systému aplikace běží, nebo třeba v jakém programovacím jazyce je psána, a především - softwarové požadavky na uživatele jsou minimální, většinou stačí právě jen webový prohlížeč. Pro uživatele to v praxi také znamená, že nemusí platit kupříkladu za drahý krabicový software, který později nakonec ani nemusí využívat. V SaaS systému takto uživate platí pouze za užívání tohoto softwaru, a to odpovídá právě tomu, jak a kdy tento software chce využívat.

PaaS

Platform as a Service (PaaS) prostředí poskytují jak infrastrukturu, tak kompletní vývojové prostředí, které umožňuje nasazení libovolných aplikací. Co je k tomuto ovšem potřeba, je aby software umístěný na tuto platformu využíval API poskytovatele, který se na druhou stranu zase stará o samotné nasazení a spouštění aplikace. Jedním z běžně užívaných poskytovatelů takovýchto cloudových služeb je např. Google se svou službou Google App Engine. Abychom mohli Google App Engine využívat, musíme např. v programovacích jazycích Python nebo Java využít odpovídajících API funkcí. Výhodou tohoto přístupu je, že je možné aplikace velmi rychle nasadit a zavést do běhu a také si dle potřeby měnit nutné prostředky pro údržbu a používání softwaru, který je ve službě umístěn. Jednou z velkých nevýhod ovšem je právě nutnost využití API v závislosti na poskytovateli. Ve chvíli co tedy takový software vytvoříme, není jednoduché jej přizpůsobit pro jiného poskytovatele, v případě, že bychom jej chtěli změnit.

IaaS

Definice *Infrastructure as a Service (IaaS)* je dle G. Reese [14] velmi závislá na poskytovateli těchto služeb, každý má totiž trochu jiný přístup k této problematice. Opět se zde projevuje různorodost, jelikož není uveden standard, který by význam IaaS diktoval. Tyto odlišnosti ovšem poskytují širší variaci, z níchž pro nás jedna může být užitečnější, než jiná. G. Reese uvádí příklady na několika nejznámějších poskytovatelích.

AWS služby jsou založeny na čisté virtualizaci. Amazon vlastní veškerý hardware, a také řídí síťovou infrastrukturu. Pokud tuto službu využijeme, máme k dispozici vše od úrovně operačního systému a výše. Můžeme žádat o virtuální instance dle potřeb a stejně tak je opět opustit, po tom, co jsme hotovi. Jedním z hlavních zaměření AWS ovšem je, že veškeré prostředky určitého serveru nejsou využity pro konkrétní virtualizaci, což nabízí jistou flexibilitu. Ovšem také nevíme, s jakou jinou aplikací potom ta naše sdílí prostředky

AppNexus poskytuje odlišný přístup k této problematice. Stejně jako AWS poskytuje přístup k serverům na požádání, AppNexus ovšem poskytuje dedikované servery s prioritou pro danou virtualizaci. To nám tedy umožňuje si dle potřeby zarezervovat libovolné množství dostupných hardwarových prostředků, což v případě AWS není možné.

2.6.2 Existující prostředky pro umístění aplikací v cloudu

Nyní se může naskytnout otázka, kterého poskytovatele cloudových služeb by pro projekt bylo vhodné zvolit. To je ovšem již záležitost správce serveru, nicméně pro orientaci v problematice si dovoluji uvést základní informace o několika největších poskytovatelích cloudových služeb. Tyto informace jsou převzaty z článku M. Neeraje [12], který se zabývá porovnáním poskytovatelů, jako jsou Amazon AWS, Google Cloud Platform a Microsoft Azure.

Amazon AWS

Amazon AWS organizuje své služby do čtyř hlavních kategorií. Výpočetní služby, databáze, počítačové sítě a úložiště a content delivery (CDN apod.). AWS poskytuje NoSQL a relační databázové služby, mnoho integrací třetích stran a silnou šifrovací platformu. Na rozdíl od Azure navíc umožňuje využití operačního systému Red Hat Enterprise Linux.

Google Cloud Platform

Google Cloud Platform se umísťuje za AWS v řadě faktorů, které jsou např. počet různých poskytovaných služeb a také geografické rozšíření jejich serverů. Google ovšem do této oblasti stále investuje, takže je možné, že tyto nedostatky budou v budoucnu odstraněny. Výhodou Google Cloud Platform je především lepší síťová architektura, oproti AWS a Azure je každá cloudová instance ve své vlastní síti. Máme také, před vytvořením této instance, dostupné silné bezpečnostní prostředky, jako jsou brány firewall, směrovače a podsítě.

Microsoft Azure

Microsoft Azure se primárně zaměřuje na služby typu PaaS. Azure se stará o automatické aktualizace platformy a vývojáři se tedy musí starat pouze o udržování svého kódu, nikoliv o bezpečnostní aktualizace a správu virtuálních strojů. Azure také nabízí integraci s dalšími službami firmy Microsoft, jako jsou Hyper-V nebo Office 365.

2.6.3 Aplikování cloudových technologií na tento projekt

Jak tedy cloud computing souvisí s tímto projektem? Celá serverová aplikace infrastruktura, která je navrhována (viz kapitola 3), je navrhována právě s myšlenkou cloudového řešení. Server se tedy nachází v cloudu, což umožňuje škálovatelnost dle počtu robotů, nebo případně je-li potřeba na straně serveru spouštět výpočetně náročné aplikace pro ulehčení výpočetní zátěže robotům.

Projekt ovšem navrhuje především samotnou aplikaci, způsob jejího zavedení již závisí na správci serveru. Serverová část je multiplatformní, ovšem systém RMS obsahuje konfigurační soubory pouze pro instalaci na OS Linux, které by ale s jistými úpravami bylo možné zprovoznit i na jiných operačních systémech. Správce serveru si tedy může zvolit, jakým způsobem server zavede, vytvoří-li si vlastní cloudový systém (což může být velmi obtížné), nebo zda využije existujících cloudových služeb, kde by se naskytovalo využití PaaS systému nějakého z existujících poskytovatelů.

Zavedení serverové aplikace v cloudu také může usnadnit Docker (viz sekce 3.3), jelikož je tento projekt dostupný také jako Docker image. V tomto případě se přímo nabízí využití cloudových služeb Docker Cloud (`https://www.docker.com/products/docker-cloud`).

2.7 Ostatní součásti

Ostatní použité technologie budou zmíněny jen okrajově, jelikož nejsou předmětem této práce, nicméně považuji za vhodné je uvést, jelikož jejich volba má na výsledek značný vliv.

Python

Python jsem zvolil kvůli možnosti multiplatformního řešení, navíc programy psané v Pythonu lze spouštět na mnoha cloudových systémech, jelikož mnoho z nich Python podporuje. Vše co je potřeba využít k implementaci projektu je v Pythonu dostupné. Umožňuje také objektově orientovaný návrh a nástroje pro tvorbu vícevláknových aplikací, stejně tak jako nástroje pro jejich synchronizaci. K tomu je možné v projektu využít implicitního modulu *Threading*, který obsahuje třídy a metody potřebné pro tvorbu nových vláken a semafory, které umožňují jejich synchronizaci a ošetření kritických sekcí při přístupu ke sdíleným datům.

Python, dle Python Software Foundation [6], je interpretovaný, objektově orientovaný, vysokoúrovňový programovací jazyk s dynamickou sémantikou. Jeho vysokoúrovňové vestavěné datové struktury v kombinaci s dynamickým typováním a dynamickou vazbou z něj dělají atraktivní jazyk pro rychlý vývoj aplikací, stejně tak jako skriptovací jazyk pro propojení již existujících komponent. Python podporuje moduly a balíky, což nabádá k tvorbě modulárního softwaru a znovuvyužívání kódu. Interpret a standardní knihovny jsou dostupné na mnoha větších platformách, což do jisté míry umožňuje přenositelnost a multiplatformnost tvořeného softwaru.

PHP

Část projektu psaná v jazyce PHP je základem uživatelského rozhraní, jelikož je v něm psán celý systém RMS. PHP v tomto kontextu pracuje jako modul na straně webového serveru. Jelikož se jedná o skriptovací jazyk, skripty psané v něm jsou interpretovány právě v tomto modulu, který slouží jako interpret jazyka, a umožňuje směrovat výstup přímo skrze webový server, čímž uskutečňuje komunikaci přímo s uživatelem. Tímto způsobem je implementována nejen logika webové aplikace, ale rovněž šablony zobrazovaných stránek, slouží tedy i k tvorbě a rozšíření statického obsahu. Protože je celý systém RMS v tomto jazyce implementován, je logické, že stejně tak bude také implementováno jeho rozšíření, které je jedním z produktů tohoto projektu. Verze, ve které je rozšíření implementováno, je PHP 5.

AJAX

Asynchronous Javascript And XML (AJAX) je vhodný pro rozšíření RMS, konkrétně pro rozhraní, které obsahuje přehled připojených robotů. To hlavně z toho důvodu, že chceme-li zobrazit stručný přehled informací, jako je stav baterie nebo odezva sítě, je vhodné, aby byl tento stav co nejaktuálnější, což ovšem již z principu HTML ani PHP neumožňuje. Je tedy nutné využít Javascript a metody pro asynchronní získávání dat.

Kapitola 3

Návrh řešení

3.1 Obecný návrh

Na robotech využívajícím platformu ROS při běžném používání běží několik serverových aplikací. V té nejzákladnější formě se jedná o roscore, často také můžeme chtít využít např. SSH serveru, nebo třeba serveru rosbridge, který je využíván pro komunikaci s webovými rozhraními, jako je třeba právě RMS. Pokud ovšem chceme k serverům vzdáleně přistupovat, je nutné vzít v úvahu pravidla firewall, NAT, směrování portů, apod. Jelikož je tento projekt cílen na roboty, lze počítat s tím, že se robot může nacházet v různou dobu v různých sítích, nebo také může využívat mobilního připojení. Takto se tedy může stát, že změna konfigurace NAT a firewall je složitá, ne-li nemožná, a i pokud změna možná je, znamenalo by to nutnost konfigurace při každém přesunu robota mezi sítěmi.

Řešení tohoto problému se nabízí několik, každý využitý protokol lze tunelovat takovým způsobem, že umístíme serverovou aplikaci na jedno zařízení a klientskou aplikaci na stranu robota. Pokud se robot spustí, klient se připojí na server (je tedy nutné znát pouze veřejnou adresu serveru, nikoliv robota) a otevřou mezi sebou komunikační tunel, kde je udržováno aktivní připojení. V případě, že se připojí např. SSH klient na server, data se přepošlou přes aktivní tunel na robota, kde se po přijetí naváže spojení se serverovou aplikací běžící na robotovi (např. SSH server). Protože se robot připojuje na server a data přeposlaná robotovi se posílají přes jeho lokální rozhraní, není třeba znát jeho veřejnou adresu.

V případě tohoto řešení se nabízí několik možností, jednou z nich je využití pouze jednoho komunikačního portu mezi serverem a robotem, pak je ale nutné odlišit, na jaký cílový port se má komunikace směrovat. Na konkrétním případu to lze popsat následovně: SSH klient se chce připojit na robota, uvažujeme-li, že server ví, na jakého robota se uživatel chce připojit a tento robot je zapnutý a běží na něm klientská část tunelovací aplikace, je tedy možné navázat spojení server – robot s využitím aktivního tunelu. V tomto případě ovšem nevíme, je-li využito pouze jednoho portu pro komunikaci server – robot, na jaký lokální port data odeslat po tom, co jsou přijata na straně robota. Problém lze samozřejmě řešit využitím jednoho portu pro každý tunel zvlášť. V tomto případě by ovšem bylo nutné na serveru nastavovat firewall pravidla zvlášť pro každý tento port, což způsobuje při každém rozšíření robotů o další komunikační protokol nutnost složitější rekonfigurace.

Pokud tedy chceme využít pouze jeden port pro tuto komunikaci, a umožnit tím jednodušší škálovatelnost robotů, je nutné zavést způsob identifikace jednotlivých protokolů. Jednou z těchto možností je přidání hlavičky, která může obsahovat informaci právě o cílovém portu, na který má být komunikace směrována. Tento způsob je využit např. v implementacích technologie VPN, kde je ke každému paketu přidána hlavička, která obsahuje

rozšiřující informace. VPN navíc nabízí možnost tunelovat připojení právě takovým způsobem, že není potřeba znát či využívat veřejnou adresu robotů pro připojení na serverové aplikace, které na nich běží. Výhodou VPN je, že existují dostupné open-source implementace této technologie, jako je např. OpenVPN, nebo SoftEther VPN. V tuto chvíli by se mohlo zdát vhodným řešením spojit roboty a uživatele v jedné VPN síti, ovšem systém ROS, konkrétně tedy roscore, neposkytuje žádnou formu zabezpečení či autentizace.

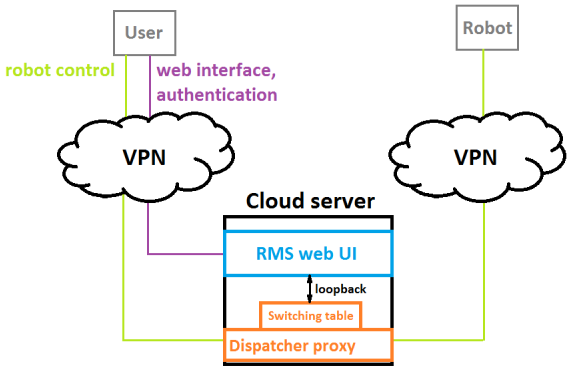
Jednou z možností by tedy v tomto případě bylo umístit na stranu robota aplikaci, která by sloužila jako vrstva mezi uživateli a roboty, a přístup k roscore zakázat pomocí firewall pravidel. Takto bychom ovšem museli stále udržovat nějakou databázi uživatelů, kteří by měli oprávnění se k robotovi připojit, nějaké další zařízení – server, je tedy v každém případě nutnou součástí. Pokud tedy již využíváme serveru, můžeme rovnou robotům tímto způsobem poskytnout výpočetní výkon, který na nich samotných nemusí být dostupný. Pokud také využijeme serveru pro autentizaci uživatelů a jejich správu, není při každé změně systému nutné měnit programové vybavení všech robotů, je zde tedy výhoda centralizace softwaru i správy uživatelů.

Jako řešení problému autorizace připojení se nabízí využití existujícího systému RMS, který nabízí nejen správu uživatelů, ale i systém rezervací, který umožňuje uživatelům si zarezervovat čas pro práci s robotem.

RMS systém je tedy spuštěn na www serveru (např. apache2) na serverovém zařízení. Na serverovém zařízení je tedy spuštěn www server, dva VPN servery a Dispatcher server. Dispatcher funguje jako prostředník spojující dvě VPN sítě, příchozí komunikace je identifikována a rozlišena dle IP adresy a data jsou přeposlána na aktuálně zvoleného robota. Cílový robot se volí v rozšíření RMS, konkrétně v rozhraní pro Dispatcher nebo automaticky při rezervaci v RMS.

Z pohledu uživatele by tedy situace mohla vypadat následovně: Uživatel chce řídit robota na vnitřní adrese X a zároveň se připojit na SSH server, který na daném robotovi běží. Uživatel se přihlásí do webového rozhraní systému RMS na adrese Y. V RMS se uživateli zobrazí seznam možných rezervací, o které může zažádat. V tuto chvíli se při pokusu o připojení přes SSH na adresu Y nestane nic a adresa X je nedostupná, protože se nachází ve vnitřní síti mezi roboty a serverem. Zvolí-li si uživatel rezervaci nějakého robota, dojde ke spojení přes Dispatcher, nyní je sice adresa Y stále nedostupná, ovšem veškerá komunikace směrovaná na adresu X je přesměrována na adresu Y pomocí Dispatcheru. SSH server běžící na robotovi X ke dostupný na adrese Y. Po rezervace RMS automaticky otevře řídící rozhraní pro uživatele, jehož hlavní částí je komunikace přes websocket, opět směrovaná na adresu Y, ovšem přesměrovaná na robota na adrese X. Komunikace RMS a Dispatcheru je detailněji popsána sekcích zaměřených na Dispatcher a rozšíření RMS.

3.2 Cloudové řešení a návrh sítě



Obrázek 3.1: Návrh sítě a komunikace

Způsobů návrhu komunikace jednotlivých prvků a robotů může být více, ovšem vzhledem k tomu, že OpenVPN nabízí vše potřebné pro tvorbu žádané infrastruktury, jsem se přiklonil k tomuto řešení. Snahou také bylo docílit cloudového řešení [14], které by umožňovalo snazší rozšířitelnost a modularitu, bez nutnosti zásahu na straně klientů a minimálním změnám na straně robotů.

Požadavky na síť jsou následující:

1. Každý klient musí mít unikátní adresu, aby nebylo nutné modifikovat klientské aplikace z důvodu jednoznačného rozlišení jednotlivých klientů.

2. Klienti musí být vzájemně izolováni, tedy jeden klient nesmí mít možnost připojit se přes tuto síť k jinému.

3. Roboti musí být vzájemně izolováni, stejně tak, jako klienti. Z toho tedy vyplývá, že i klienti jsou izolováni od robotů.

4. Spojení mezi klientem a robotem může být povoleno pouze oprávněným uživatelům.

5. Adresový prostor robotů by bylo vhodné oddělit od adresového prostoru klientů.

Lze tedy vytvořit dvě VPN sítě, přes které se bude možné připojit k serveru. To nám také umožní jednoduše využívat aplikace pro složitější výpočty, které mohou být spuštěny na serveru a od robotů přijímat např. pouze sensorická data.

Prostředníkem, který tyto sítě vzájemně propojuje je Dispatcher, který má vlastní směrovací tabulku, podle které přeposílá veškeré požadavky na zvolené síťové porty na robota, se kterým je uživatel dle této tabulky spojen (jak bylo již dříve zmíněno, dle páru IP adres). Záznamy v tabulce je možné dynamicky měnit pomocí jednoduchého komunikačního protokolu, ideálně pouze přes lokální síťové rozhraní, jelikož protokol sám o sobě není nijak zabezpečen.

Rozšíření systému RMS tento protokol Dispatcheru využívá. Vzhledem k tomu, že RMS je vytvořen v jazyce PHP, všechny požadavky na spojení přicházejí ze serveru samotného. Tedy komunikace RMS–Dispatcher probíhá na lokálním rozhraní, pokud je web server i Dispatcher spuštěn na stejném zařízení.
Celé schéma sítě a připojení je znázorněno na obrázku 3.1.

3.2.1 Směrování TCP a UDP komunikace

TCP

Směrování TCP komunikace je relativně jednoduché. Na serveru je potřeba otevřít socket, který naslouchá na rozhraní pro příjem komunikace od uživatelů, zatímco na robotech naslouchá nějaká aplikace na libovolném aplikačním protokolu. Po přijetí připojení na straně serveru se pak dle pravidel (např. dle směrovací tabulky) určí, na jakého robota má být otevřeno nové připojení. Poté dojde ke spojení přijímajícího serverového socketu s tímto nově vytvořeným. Dále se vytvoří nové vlákno nebo proces, který má na starost komunikaci opačným směrem. Vzniknou tedy dvě vlákna nebo procesy, které obsluhují danou komunikaci klienta a robota. Pro každé TCP spojení pak vzniká pár těchto spojení.

UDP

S UDP komunikací je to složitější, jelikož nemůžeme bez zapouzdření či úprav jednoduše komunikaci zprovoznit obousměrně. Jednou z možností je udržovat tabulku aktivních připojení a portů, přes které komunikace probíhá a jaké porty byly zarezervovány pro zpětnou komunikaci. Tím ovšem dochází k problému, jelikož UDP spojení se nijak neudržuje, že by se nám tabulka začala plnit a bylo by nutné implementovat nějakou formu udržování této tabulky, např. odstranění spojení z tabulky po jisté době neaktivity. Další možností je zapouzdření, ovšem to by vyžadovalo síťový prvek či aplikaci na rozbalení zapouzdřených paketů na straně serveru, tedy další část, která by roboty zatěžovala.

3.3 Zavedení a distribuce

Tento projekt je open-source aplikace, z toho důvodu jsou tedy všechny zdrojové kódy umístěny na službě GitHub. Součástí umístění na GitHub jsou také readme soubory, které obsahují dokumentaci k projektu a návod k instalaci a použití. Soubory jsou umístěny v repozitáři na adrese `https://www.github.com/mjezersky/robotcloudserver`.

Kromě umístění na službě GitHub je serverová část také k dispozici prostřednictvím služby Docker, jako tzv. Docker Image. Tento Image je dostupný ke stažení skrze `mjezersky/robotcloudserver`. Podrobnější návod k instalaci a zavedení tohoto Docker Image lze nalézd v readme v GitHub repozitáři tohoto projektu.

Docker

Docker je open-source projekt, který umožňuje automatizaci zavádění softwaru do provozu pomocí virtualizace a tzv. kontejnerů. Virtualizace Dockeru se liší od ostatních běžných virtualizačních nástrojů jako je VMWare, VirtualBox apod. způsobem, kterým virtualizaci provádí. Konkrétní popis funkce Dockeru nebudu rozvádět, jelikož není předmětem této práce, nicméně Docker tímto způsobem dosahuje minimalistických rozměrů jednotlivých virtuálních aplikací.

Kapitola 4

Implementace

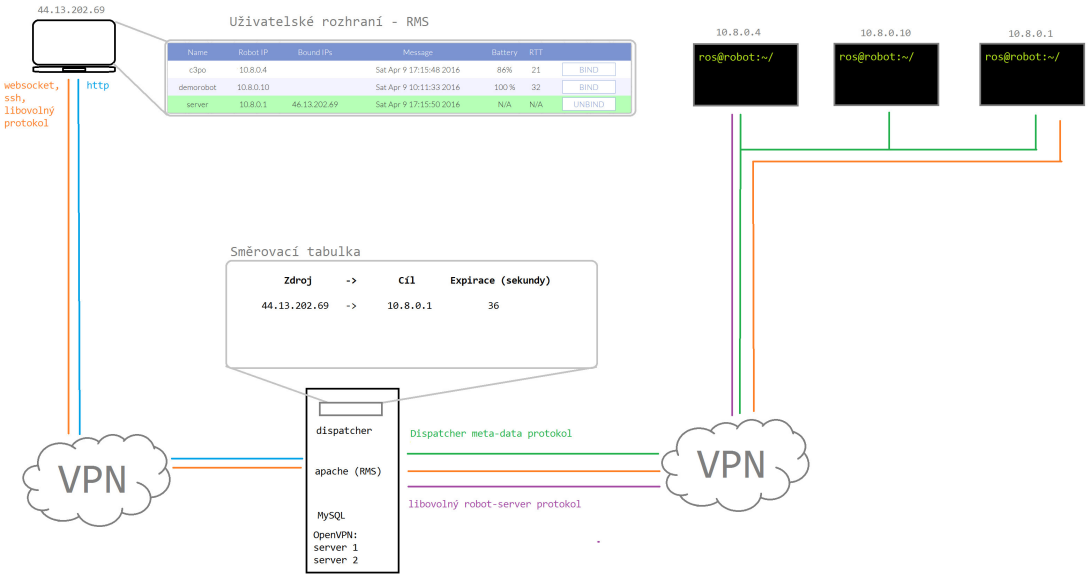
Projekt byl programově implementován v jazycích Python (Dispatcher) a PHP (rozšíření RMS). V této kapitole budou zmiňovány tři základní prvky: uživatel, server (zařízení, na kterém běží webový server a Dispatcher) a robot (zařízení, na kterém běží systém ROS, tedy *roscore*).

Na serveru je pro správnou funkci nutné mít spuštěný OpenVPN server, optimálně se dvěma serverovými konfiguracemi – jednou pro roboty a jednou pro uživatele. Tím zajistíme, že pokud Dispatcher naslouchá na rozhraní pro roboty, nemůže se stát, že by se nějakému uživateli povedlo se do Dispatcheru zaregistrovat jako robot. Tato konfigurace samozřejmě není nutná, nicméně je důrazně doporučována. Detailnější popis implementace jednotlivých součástí lze nalézt v následujících sekcích.

4.0.1 Dispatcher

Dispatcher je psán v jazyce Python, který umožňuje objektově orientovaný návrh, toho je tedy využito a tento návrh je také dodržen, pro snadnější budoucí úpravy, větší modulárnost a přehlednost kódu. Jedná se o vícevláknový program, kde každý síťový prvek (socket server/klient) běží ve vlastním vláknu. Ošetření kritických sekcí je zajištěno semafory dostupnými v modulu `Threading`. Hlavní část Dispatcheru je server, který i sám o sobě funguje jako plnohodnotný reverzní proxy server, nicméně Dispatcher má a klientskou část, která umožňuje odesílání metadat o klientech na server, pro vytvoření přehledu a seznamu připojených zařízení.

Komunikace přes Dispatcher je znázorněna v diagramu 4.1.



Obrázek 4.1: Příklad možného spojení přes Dispatcher.

Server

Po spuštění se nejprve inicializuje server pro konfiguraci směrovací tabulky a přijímání informací od robotů a také server pro příchozí komunikaci od uživatelů a tvorbu spojení mezi uživateli a roboty. Takovéto nové spojení vzniká v případě, že se nějaký uživatel připojí na jeden z portů, které byly zařazeny do seznamu směrovaných portů a zároveň existuje záznam o směrování jeho adresy ve směrovací tabulce. Veškerou potřebnou konfiguraci je možné provést ve skriptu `launch_server.py`.

V Dispatcheru je poté možné vytvořit několik typů směrování, tedy např. že všechna příchozí komunikace na port 100 od klientů, bude směrována na port 80 vybraného robota. V samotném Python skriptu by přibyl řádek `disp.addTunnel(100, 80)`.

Výchozí protokol pro tyto tunely je TCP, UDP tunel vytvoříme jednoduše nastavením nepovinného argumentu `udp` na logickou 1 (tedy True), směrování UDP portu 80 na port 100 by tedy vypadalo následovně:
`disp.addTunnel(100, 80, udp=True)`

Server socket pro přijímání dat od robotů a konfiguraci směrovací tabulky na straně serveru funguje tak, že nejprve přijímá data od připojených socketů a podle přijaté zprávy rozhodne, zda se jedná o robota (`TUNNEL_CLIENT`) nebo o konfigurační protokol (`APP_CLIENT`).

Jedná-li se o robota, server se periodicky dotazuje na aktuální data (výchozí perioda je 1 vteřina) všech připojených robotů (Dispatcher klientů). Jedná-li se o konfigurační protokol, server očekává zprávu, jako např. žádost o spojení IP klienta a IP robota, třeba žádost o spojení klienta s adresou 10.8.0.2 s robotem s adresou 10.8.0.5 na dobu 60 vteřin, by vypadala následovně:
`B10.8.0.2#10.8.0.5#60`
Ve výchozím nastavení se po vypršení této doby, nebo při změně propojení IP adres, přeruší veškerá aktivní spojení mezi původním párem adres. Pokud bychom tomu chtěli zabránit,

je možné inicializační metodě Dispatcheru předat nepovinný argument `interruptOnRebind=False`.

Dispatcher také pomocí třídy `Collector` sbírá veškeré informace periodicky získávané od robotů a vkládá si je do vlastní struktury, pro pozdější využití, jako např. při odesílání seznamu připojených robotů a jejich dat do webového uživatelského rozhraní. Metody této třídy slouží ke snadné manipulaci s daty, ke kterým přistupujeme asynchronně, k tomu metody této třídy využívají několika semaforů.

Round Trip Time se počítá na straně serveru tak, že server odešle robotovi zprávu `ECHO`, a poté měří čas odpovědi robota stejnou zprávou. Nebyl využit ICMP ping, protože pro tvorbu ICMP zprávy nemá běžný uživatel v systému Ubuntu oprávnění.

Vzhledem k tomu, že veškerá komunikace probíhá asynchronně, je zde využito semaforů pro zajištění synchronizace dat při přístupu k datům sdíleným mezi vlákny.

Klient

Na robotech je pro správnou funkci celé aplikace nutné mít spuštěný minimálně Dispatcher klient, který odesílá informace na Dispatcher server, jako je volitelná zpráva, stav baterie a odezva, a to i když není s žádným klientem svázán ve směrovací tabulce Dispatcher serveru. Tato data slouží pro zobrazení obecného přehledu všech robotů, proto jsou odesílány právě informace, jako stav baterie nebo odezva (Round Trip Time) [10], o kterých je vhodné mít obecný přehled. Dispatcher server sám o sobě není na klientu pro jeho správnou funkčnost závislý, ovšem v případě tohoto projektu, rozšíření RMS, konkrétně tedy webové rozhraní pro řízení Dispatcheru, využívá data přijatá z Disptacher klientů k zobrazení přehledu a seznamu robotů, ke kterým se lze připojit, můžeme tedy pouze požádat o svázání s robotem, který se nachází v této tabulce. Nicméně i pokud by na robotovi Dispatcher klient spuštěn nebyl, automatické svázání adres stále bude fungovat skrze otevření rozhraní z RMS při zarezervování práce s robotem nebo při pokračování ve stávající rezervaci.
Důvod, že bude toto spojení fungovat je ten, že Dispatcher server potřebuje informaci o adrese uživatele a adrese robota, pokud nevíme, jakou má robot adresu v síti VPN (není v tabulce webového rozhraní Dispatcheru), nemůžeme je spojit ručně, ovšem při rezervaci máme informaci o *Environment* robota, což jeho adresu obsahuje.

Klient po spuštění pracuje jiným způsobem v závislosti na tom, je-li využito *ROS Topic* k získávání informací o stavu baterie. Je-li nějaký *ROS Topic* určen, a také je-li k dispozici *rospy* modul, dojde nejprve k pokusu o připojení na *roscore* a spuštění přijímání zpráv na zvoleném *Topic*. Stav baterie je pak při každém přijetí zprávy ukládán do proměnné, která je asynchronně čtena ze strany hlavního vlákna klienta. Jelikož klient tyto data čte vždy na požádání serveru, ve výchozí situaci k tomu tedy dochází jednou za vteřinu, není tedy zajištěna synchronizace čtení a zápisu do proměnné o stavu baterie, jelikož nám nevadí, když omylem dostaneme trochu starší informaci, protože baterie stav baterie se za jednu vteřinu zásadně nezmění.

Pokud není stanoven *Topic* pro získání informací o baterii, je využito souboru `/sys/class/power_supply/BAT0/capacity`, pokud tedy tento soubor existuje, pokud ne, je stav baterie uveden jako `N/A`.

Veškerá potřebná konfigurace Dispatcher klient je obsažena ve spouštěcím souboru `launch_client.py`. Více informací je také uvedeno v readme souborech v GitHub repozitáři. Klient vyžaduje zvolení názvu robota, ve výchozím nastavení je to systémová proměnná `hostname`. Tento název ovšem může být jakýkoliv, jediná podmínka je, aby byl unikátní

mezi všemi roboty připojenými na Dispatcher server, jinak bude spojení odmítnuto.

Po spuštění se klient pokouší o připojení na Dispatcher server, pokud se mu spojení povede navázat, čeká na žádost serveru. Odpovídá na dvě možné zprávy, `ECHO` (měření RTT) a `DISPATCHER_DATA_REQUEST` (sbírání dat o robotech).

Tento klient je velmi malý program, který zatěžuje roboty pouze minimálně.

Konfigurační protokol

Dispatcher umožňuje dynamickou konfiguraci pomocí jednoduchého síťového protokolu. Server po navázání spojení nejprve odešle uvítací zprávu `HELLO` a očekává řetězec o typu protokolu, chceme-li tedy měnit konfiguraci, odešleme řetězec `APP_CLIENT`. Po odeslání této zprávy server odešle zpět řetězec `ACK`, poté již veškerá komunikace probíhá dle následujících pravidel. Každá zpráva se odesílá na Dispatcher server v takové formě, že se nejprve odešle délka následující zprávy, poté znak `#` a za ním následuje zpráva samotná. Chceme-li tedy získat data ze směrovací tabulky, chceme tedy odeslat zprávu `BINDINGS`, odešleme následující řetězec `8#BINDINGS`. Zprávy a jejich význam jsou popsány v tabulce 4.1.

Zpráva	**Popis**
`BINDINGS`	vrátí seznam všech svázaných adres (obsah směrovací tabulky)
`GET_ALL_DATA`	vrací JSON objekt, který obsahuje veškeré informace o připojených robotech, tedy název (ID) robota, stav baterie, RTT a volitelnou zprávu, dále také obsahuje seznam všech svázaných adres (obsah směrovací tabulky)
`B`*userip*`#`*robotip*`#`*time*	modifikuje záznam o svázaných adresách ve směrovací tabulce, konkrétně tedy na kterého robota (adresa *robotip*) má být komunikace od uživatele (adresa *userip*) směrována, a také po jaké době v sekundách (*time*) platnost této informace vyprší.
`G`*robotid*	vrací informace o robotovi s názvem (ID) *robotid*. Jedná se o stejné informace jako z `GET_ALL_DATA`, ovšem pouze pro zvoleného robota.

Tabulka 4.1: Zprávy konfiguračního protokolu Dispatcheru.

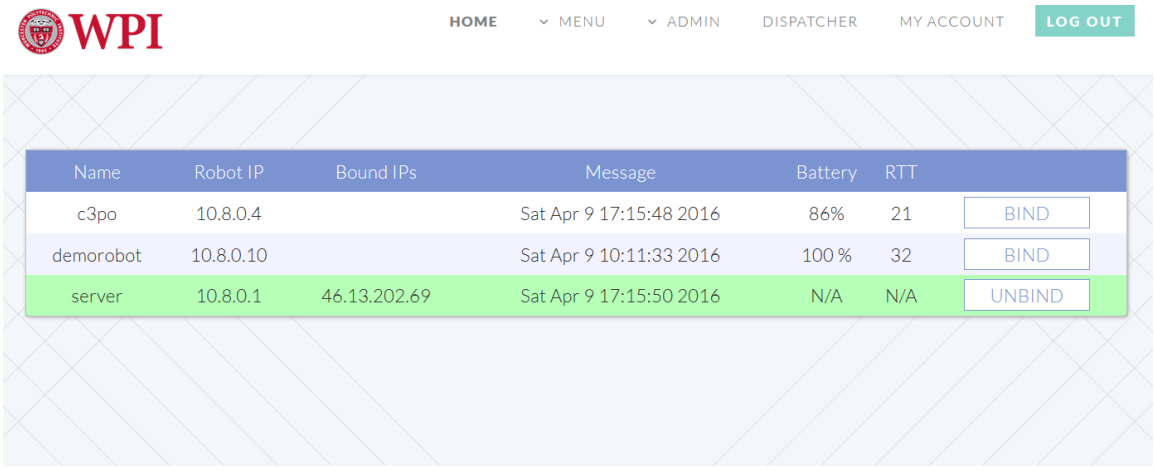
4.0.2 Rozšíření RMS

Celý systém bylo nutné poupravit, jelikož způsob spojení přes Dispatcher je lehce odlišný od toho přímého, v uživatelském rozhraní se to projevilo především tím, že není možné zjistit stav konkrétní serverové aplikace spuštěné na robotech, pokud se nejedná o toho robota, ke kterému je uživatel právě připojen.

Hlavní změnou ovšem bylo zakomponování protokolu a řízení Dispatcheru do rezervačního systému RMS. Pokud si tedy uživatel zažádá o spuštění řídícího rozhraní robota, kterého má právě zarezervovaného, dojde k vytvoření zprávy o IP adrese klienta, IP adrese

robota a době rezervace, která se odešle Dispatcheru, aby bylo možné navázat spojení. Pokud má uživatel platnou rezervaci, je možné provést svázání IP adres ručně, přes webové rozhraní samotného Dispatcheru. Pokud je přihlášený uživatel administrátorem, může provést svázání adres i bez rezervace (viz obrázek 4.2). Toto svázání může uživatel chtít provést např. pokud nechce robota řídit, ale pouze se připojit k SSH serveru, který je na něm spuštěn. Dále bylo v RMS také nutné změnit způsob připojování v rozhraní, kde se řídící rozhraní nepřipojuje přímo na adresu robota, ale na adresu serveru, který zajistí přesměrování.

Řídící rozhraní vyžaduje spuštění rosbridge serveru na straně robota, aby systém RMS mohl komunikovat s platformou ROS pomocí JSON objektů. Tato komunikace je dalším důvodem, proč nestačí pouze VPN mezi roboty a serverem bez využití Dispatcheru. Nebylo by totiž možné, přes websocket vytvořený v javascriptu, se na robota připojit.



Obrázek 4.2: Webové uživatelské rozhraní Dispatcheru s přehledem připojených robotů.

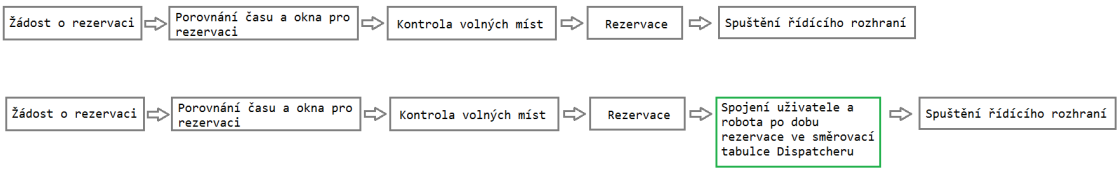
Rozhraní Dispatcheru

Součástí rozšíření je grafické rozhraní pro konfiguraci směrovací tabulky Dispatcheru (viz obrázek 4.2). Toto rozhraní je implementováno ve třech částech – AJAX části, která slouží k periodickému získávání dat (jednou za vteřinu) a udržování aktuálního obsahu na stránce, javascript logice, která zajišťuje zobrazení správných ovládacích prvků a přehledu, a také PHP částí, která umožňuje konfiguraci samotného Dispatcheru. Pokud by se stalo, že by uživatel neměl oprávnění ke spojení s vybraným robotem, ale upravil by si javascript kód ručně tak, aby poskytoval tlačítko pro připojení, nic by se nestalo, jelikož v PHP dochází k ověření platnosti žádosti.

Systém rezervací

Nutnou úpravou si také prošel rezervační systém, jelikož do něj byla zakomponována žádost o spojení s vybraným robotem (viz obrázek 4.3). Do *Controller* části systému rezervace bylo doplněno požádání o spojení uživatele a robota skrze Dispatcher. Cílová adresa robota je určena z *Environment*, které se váže na danou rezervaci, stejně tak jako čas, který je Dispatcheru předán. Čas se určuje právě po zarezervování časového okna v RMS, a to z rozdílu času konce rezervace a aktuálního času na serveru. Podobným způsobem také probíhá opětovné svázání, pokud se připojujeme na již probíhající rezervaci, což také umožňuje

libovolné přepínání mezi aktuálně zarezervovanými roboty.



Obrázek 4.3: Diagram činnosti při rezervaci.

Přehled rosbridge a MJPEG serverů

Jedna z funkcí, kterou RMS umožňoval, bylo zobrazit přehled rosbridge a MJPEG serverů včetně jejich stavu. Jelikož se k ověření dostupnosti ale nedochází na straně serveru, nýbrž na straně klientů, tato funkce je dostupná pouze pro jeden jediný rosbridge/MJPEG server, a to na adrese, která je přes Dispatcher právě svázána s uživatelem. Bylo by jistě možné, po dalších úpravách rozhraní, vytvořit statickou stránku, která by alespoň jednou při načtení (nebo při každém AJAX dotazu) poskytovala stav serverů, který by byl získáván např. přes komunikační protokol Dispatcheru, nebo přímo na straně serveru, pouhým testováním dostupnosti všech připojených robotů. Stávající implementace ovšem dovoluje zobrazit stav rosbridge/MJPEG serveru na aktuálně zvoleném robotovi a zároveň aktuální stav připojení, baterie a dalších vlastností skrze rozhraní Dispatcheru, proto jsem zachování seznamu rosbridge/MJPEG serverů nepovažoval za důležité.

4.0.3 Dispatcher jako samostatná jednotka

Dispatcher je, jak bylo dříve zmíněno, možné použít i samostatně pro jiné účely, než jen řízení připojení k robotům. Jedná se o plnohodnotný reverzní proxy server, který je nezávislý i na své klientské části, je-li to potřeba. Jediným prvkem, který vyžaduje, je libovolná aplikace, která jej bude řídit pomocí jeho konfiguračního protokolu. Bylo by tedy i možné z něj udělat statický prvek, který by pouze převedl konfigurační soubory do zpráv tohoto protokolu.

Kapitola 5

Testování, experimenty a měření

5.1 Testování

Projekt byl testován především na virtuálních počítačích a robotech v laboratoři. Testování probíhalo spouštěním serveru a klienta v různých konfiguracích, tzn. různé konfigurační (launch) soubory. Server byl spouštěn především z Docker image, což umožňovalo snadný návrat do výchozího nastavení a stavu, bylo-li to potřeba. K testování spojení byla tedy vytvořena sada konfiguračních souborů a také implementovány jednoduché aplikaci simulující síťový provoz a měření rychlosti – echo server s ověřením ID klienta (pro ověření správnosti směrování), jednoduchý http server a klient-server pro odesílání libovolné velikosti náhodných dat. Na závěr bylo spojení testováno i na aplikacích třetích stran, jako je SSH klient-server. VLC stream, apod. Kromě spojení byl také testován rezervační systém rozšíření RMS a všechny ostatní provedené změny.

5.2 Experimenty a měření

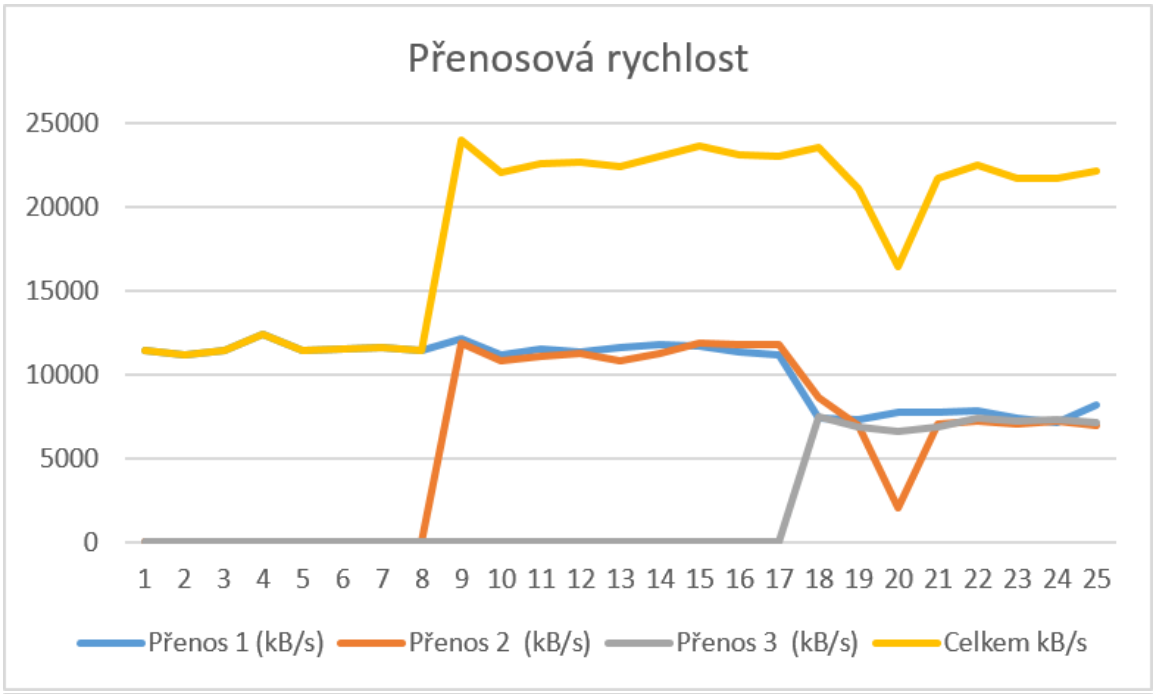
Kromě testování byla také provedena sada měření, konkrétně závislost připojených uživatelů a robotů na zátěž paměti, procesoru a sítě na klienta i serveru.

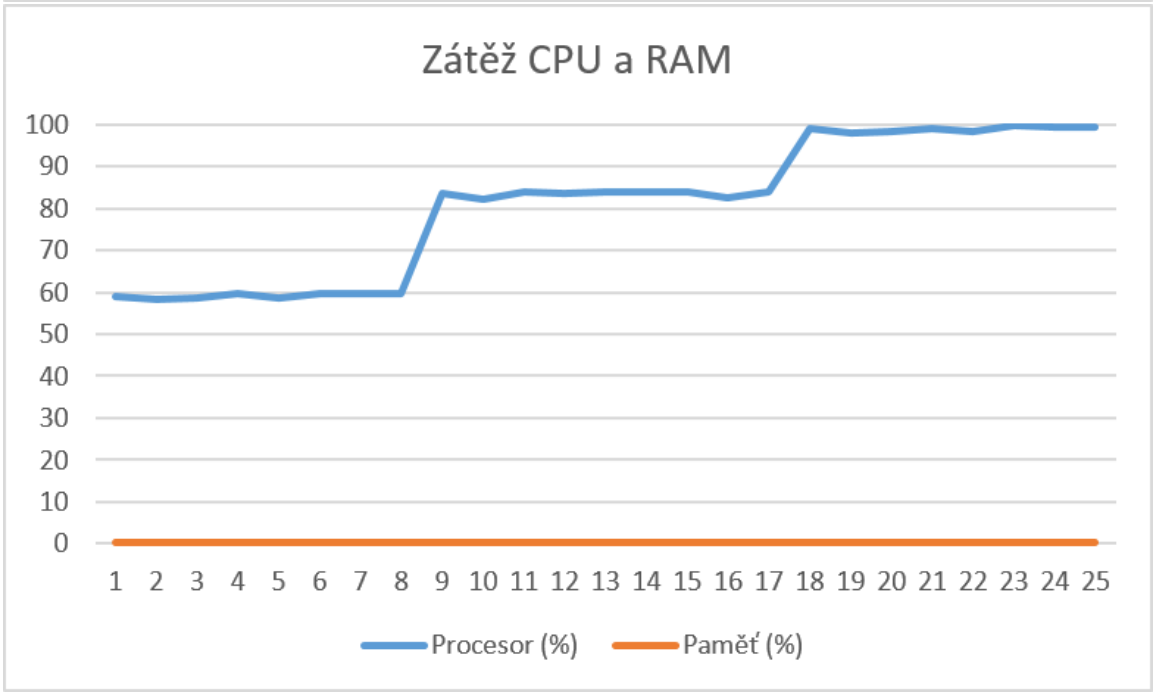
Ze strany klienta je zátěž minimální, vliv Dispatcher klienta je téměř zanedbatelný, jediné, co je skutečně ovlivněno je zátěž sítě, která je sice způsobena ostatními aplikacemi, ovšem vzhledem k využití OpenVPN přidáváme ke každému paketu OpenVPN hlavičku. Vliv OpenVPN zapouzdření byl měřen pomocí programu Wireshark a bylo zjištěno, že největší možná hlavička, která byla naměřena, je 69 bytů na paket. Při některých konfiguracích byla pouze 26 bytů, a závisí především na využitých šifrovacích metodách. Zpoždění způsobené Dispatcherem při směrování dat bylo měřeno až na pěti uživatelích a pěti robotech. Toto zpoždění lze považovat při takovém počtu robotů a uživatelů za zanedbatelné (<1ms) pro běžné použití.

Měření probíhalo na stroji s dvoujádrovým procesorem Intel Celeron Dual-Core o frekvenci 1.5GHz a 2GB operační paměti.

Maximální dosažená přenosová rychlost Dispatcheru, naměřená přes místní síť, byla okolo 24 MB/s, tedy okolo 192 Mbit/s. Poté byla rychlost omezena zatížením procesoru. V porovnání s OpenVPN, Dispatcher zatěžuje procesor zhruba o čtvrtinu méně, v závislosti na zpracovávané síťové komunikaci.

Měření zobrazená v grafech 5.1 se týkají pouze zátěži Dispatcherem, nikoliv OpenVPN.





Obrázek 5.1: Grafy zobrazují závislost objemu přenesených dat přes Dispatcher na zátěž procesoru a operační paměti.

Kapitola 6

Závěr

V tomto textu byl zmíněn návrh a implementace cloudového serveru, který umožňuje řízení a správu robotů přes webové rozhraní, nebo přes libovolné aplikace vytvořené vývojáři. Byla stanovena architektura sítě potřebná pro tuto implementaci a způsob její realizace.

Projekt poskytuje řešení problému s jednoduchým a centralizovaným řízením robotů, které zároveň umožňuje snadnou rozšiřitelnost a flexibilitu. To vše je ve výsledku součástí jednoho softwarového balíku, který umožňuje snadné sestavení této infrastruktury a řeší problém, který žádný software nebo softwarový balík běžně dostupný v open-source a ROS komunitě zatím neřeší.

Přínosem tohoto projektu může být také samostatné využití jedné z jeho komponent, Dispatcheru, který je na ostatních součástech nezávislý, a lze jej použít i v jiných projektech, kde by bylo potřeba využít reverzní proxy server, který nabízí dynamickou rekonfiguraci.

Projekt je umístěn na službě GitHub, kde na něj mohou navázat další vývojáři, případně využít jeho libovolnou část ve svých projektech. Stejně tak je také možné získat Docker image, který lze stáhnout pomocí identifikátoru `mjezersky/robotcloudserver`. Zároveň byla podána žádost o umístění na stránkách projektu ROS (wiki.ros.org).

Možností v pokračování na tomto projektu je několik, buďto možná snaha o zdokonalení samotného Dispatcheru, např. zakomponováním VPN protokolu přímo do něj, nebo jiné řešení, které by jeho činnost zefektivnilo, nebo také zdokonalení a rozšíření uživatelského rozhraní RMS. Sám o sobě tento projekt neposkytuje komplexní řídící rozhraní pro roboty, pouze demonstrační rozhraní pro pohyb a příjem obrazu, jelikož se zabývá především infrastrukturou propojení a nabízí platformu, na které lze rozhraní vytvářet. Dále by také bylo možné rozšířit stávající správu uživatelů v systému RMS, např. o uživatelské skupiny, apod.

Článek o této práci je také zveřejněn ve sborníku studentské konference Excel@FIT 2016 (http://excel.fit.vutbr.cz/submissions/2016/021/21.pdf) pod identifikačním číslem 21.

Literatura

[1] *The Robot Management System*. ROS Wiki [online]. [cit. 2016-04-08]. Dostupné z: http://wiki.ros/rms.

[2] *Rosbridge*. ROS Wiki [online]. [cit. 2016-04-08]. Dostupné z: http://wiki.ros/rosbridge_suite.

[3] *The Standard ROS JavaScript Library*. ROS Wiki [online]. [cit. 2016-04-08]. Dostupné z: http://wiki.ros/roslibjs.

[4] *VPN Server behind the firewall*. [obr.] Microsoft TechNet [online]. [cit. 2016-05-11]. Dostupné z: https://technet.microsoft.com/en-us/library/cc753364%28v=ws.10%29.aspx.

[5] *What is Cloud Computing*. Amazon Web Services [online]. [cit. 2016-05-05]. Dostupné z: http://aws.amazon.com/what-is-cloud-computing/.

[6] *What is Python? Executive Summary*. Python Software Foundation [online]. [cit. 2016-05-07]. Dostupné z: https://www.python.org/doc/essays/blurb/.

[7] *What is ros*. ROS Wiki [online]. [cit. 2016-04-26]. Dostupné z: http://wiki.ros/introduction.

[8] *Virtual Private Networking: An Overview*. Microsoft TechNet [online], Sep 04, 2011. [cit. 2016-05-05]. Dostupné z: https://technet.microsoft.com/en-us/library/bb742566.aspx.

[9] Crick, C. *Rosbridge: Ros for non-ros users*. Proceedings of the 15th International Symposium on Robotics Research, 2011.

[10] Dovrolis, C.; Jihang, H. Passive estimation of TCP round-trip times. 2002. ISSN 01464833, 10.1145/571697.571725.

[11] Luotonen, A.; Altis, K. Selected Papers of the First World-Wide Web Conference World-Wide Web proxies. 1994, 27, 2. ISSN 0169-7552.

[12] Neeraj, M. *Amazon AWS vs Google Cloud Platform vs Microsoft Azure: Which Public Cloud Is Best for You?* DAZEINFO [online], rev. 2004-11-11. [cit. 2016-05-06]. Dostupné z: http://dazeinfo.com/2015/05/22/amazon-aws-google-cloud-microsoft-azure/.

[13] Quigley, M. *ROS: an open-source Robot Operating System*. ICRA workshop on open source software, 2009.

[14] Reese, G. *Cloud application architectures*. O'Reilly Media, Inc., 2009. ISBN 9780596555481.

[15] Srisuresh, P.; Holdrege, M. : IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663, IETF.org [online], 1999, [cit. 2016-05-05]. Dostupné z: https://tools.ietf.org/html/rfc2663.

[16] Villanueva, J. C. *Forward Proxy vs Reverse Proxy*. jscape.com [online], Aug 06, 2012. [cit. 2016-05-5]. Dostupné z: http://www.jscape.com/blog/bid/87783/Forward-Proxy-vs-Reverse-Proxy.

Přílohy

Seznam příloh

Příloha A

Obsah DVD

demo_konfigurace/	předchystané soubory pro demonstraci
latex/	zdrojové soubory pro sestavení tohoto textu v LaTeXu
robotcloudserver/	zdrojové soubory a návody - git repozitář projektu
plakat.pdf	plakát ve formátu PDF
readme.txt	manuál (a odkazy na návody) k instalaci a použití
robotcloudserver.tar	uložený docker image ve formátu TAR, lze načíst pomocí `docker load -i robotcloudserver.tar`
text.pdf	tento text práce ve formátu PDF

Příloha B

Manuál

Veškeré informace potřebné k instalaci, spuštění a používání projektu jsou dostupné na DVD (viz soubor readme.txt v kořenovém adresáři DVD) nebo v GitHub repozitáři (`https://www.github.com/mjezersky/robotcloudserver.git`).

Příloha C

Plakát

Plakát je dostupný na DVD (soubor plakat.pdf v kořenovém adresáři DVD) nebo ve sborníku studentské konference Excel@FIT 2016 (http://excel.fit.vutbr.cz/submissions/2016/021/21_poster.pdf).