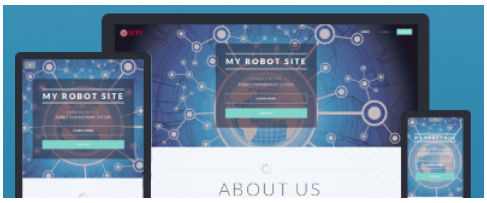
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

Cloudový server pro správu robotů

Matouš Jezerský*

ROS



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 přímého VPN připojení, které by exponovalo nezabezpečený systém ROS, a také s důrazem na co nejmenší softwarové požadavky na uživatele a roboty. Výsledkem je open-source aplikace, která toto umožňuje.

Problém je řešen vytvořením serveru, který jedná jako prostředník mezi klientskými aplikacemi a servery běžícími na robotech (dále Dispatcher). Dispatcher funguje jako reverzní proxy server, který umožňuje dynamickou rekonfiguraci za běhu, a to pomocí síťového rozhraní a jednoduchého komunikačního protokolu. Spojení klientů – serveru a robotů – serveru je zajištěno pomocí OpenVPN sítí izolujících jejich klienty. 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.

Dispatcher sám o sobě je využitelný i pro jiné účely, než je spojení uživatelů a robotů, vzhledem k tomu, že Dispatcher je nezávislý na síťové aplikaci či protokolu na aplikační vrstvě sítě (dle TCP/IP modelu), kterou zprostředkovává. Konfigurace Dispatcher serveru je snadno dynamicky přenastavitelná s pomocí jednoduchého síťového protokolu.

Software bude dostupný skrze službu *GitHub* a serverová část také jako *Docker image*.

Klíčová slova: Robot Management — Reverse Proxy — Dispatcher — Robot Cloud

Přiložené materiály: Github repozitář — Docker image — Robot Management System

*xjezer01@stud.fit.vutbr.cz, *Fakulta Informačních Technologií, Vysoké Učení Technické v Brně*

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. Za robota je v tomto článku považováno zařízení s platformou ROS [1].

Navrhovaná varianta řešení se skládá ze tří základních částí - sítě s využitím VPN, Dispatcheru a RMS [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, sítě firewall, apod. Dis-

patcher 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 [3], 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.

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. 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.

2. ROS a RMS

Projekt je cílen na roboty, na kterých je využita platforma **Robotic Operating System (ROS)**. Jedná se o framework pro zjednodušení tvorby programů a aplikací zaměřených na robotiku. ROS je sada nástrojů, knihoven a konvencí, standardně využívaná na operačním systému Ubuntu. 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*).

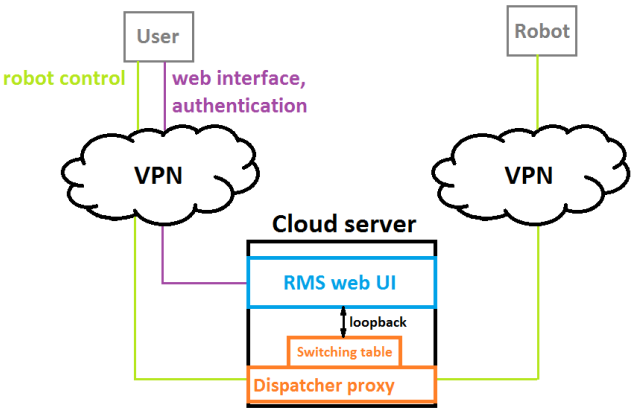
V tomto projektu je využito také systému **Robot Management System (RMS)** [2], který poskytuje nástroje k tvorbě a používání webových rozhraní pro řízení robotů, přijímání obrazových dat, apod. Tento systém zajišťuje autentizaci uživatelů a správu rozhraní k ovládání robotů, a také poskytuje rezervační systém, kde je možné mezi uživateli rozvrhnout čas pro práci s roboty. Ovládací rozhraní umožňuje řízení robotů se systémem ROS pomocí javascript knihovny *roslibjs* [4], které ke komunikaci využívá JSON objekty a *rosbridge* server [5] na straně robota. *Rosbridge* převádí JSON objekty na ROS zprávy a naopak.

3. Cloudové řešení a návrh sítě

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 [6], které by umožňovalo snazší rozšiřitelnost 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.



Obrázek 1. Návrh sítě a komunikace

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 1.

4. Implementace

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

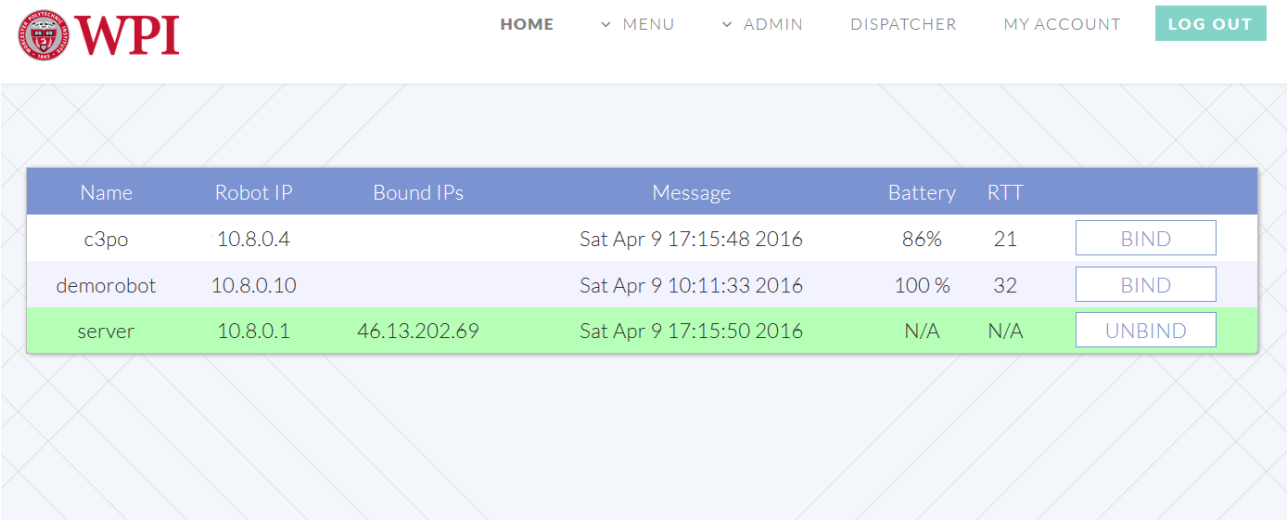
Na serveru je pro správnou funkci nutné mít spuštěný OpenVPN server se dvěmi serverovými konfiguracemi – jednou pro roboty a jednou pro klienty. Tím zajistíme, že pokud Dispatcher naslouchá na rozhraní pro roboty, nemůže se stát, že by se nějakému uživateli povedlo se do Dispatcheru zaregistrovat jako robot.

4.1 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.

Po spuštění se nejprve inicializuje server pro konfiguraci směrovací tabulky a přijímání informací od robotů a také server pro příchozí komunikaci od klientů a tvorbu tzv. tunelů. Nový tunel vzniká v případě, že se nějaký klient připojí na jeden z portů, které byly zařazeny do seznamu směrovaných portů.
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, směrování UDP portu 80 na port 100 by tedy vypadalo následovně:
`disp.addTunnel(100, 80, udp=True)`
Je také potřeba dodat, že tunel se vytvoří pouze v případě, že ve směrovací tabulce existuje záznam pro spojení IP adresy klienta a IP adresy robota.

Na robotech je nutné mít spuštěný minimálně Dispatcher klient, který odesílá Dispatcher serveru informace, jako je stav baterie a volitelná zpráva, a to i když není s žádným klientem svázán ve směrovací tabulce Dispatcher serveru. Tato data slouží pro zobrazení obecného přehledu všech robotů, jako je právě stav baterie nebo odezva (Round Trip Time) [7]. Tento klient je velmi malý program, který zatěžuje roboty pouze minimálně. Server socket pro přijímání dat od robotů a konfiguraci směrovací tabulky na straně serveru fun-



Obrázek 2. Webové uživatelské rozhraní Dispatcheru s přehledem připojených robotů.

guje 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`).
Stav baterie lze zjistit buď pomocí libovolného *Topic*, nebo ze souboru
`/sys/class/power_supply/BAT0/capacity`

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.

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 metody `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.
Vzhledem k tomu, že veškerá komunikace probíhá asynchronně, je zde využito semaforů pro zajištění správnosti dat v kritických sekcích.

4.2 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 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.

5. Návrh testování

Projekt byl zatím testován na virtuálních počítačích a robotech v laboratoři, ovšem pouze velmi základní testovací sadou. Konrétně testování spojení mezi klien-

tem a vybraným robotem pomocí jednoduchého echo serveru s ověřováním, zda se jedná o skutečně požadovanou adresu.

Za vhodné považuji vytvořit sadu testů pomocí automatizovaných Travis CI testů, které budou využitelné i pro případné zájemce v rozšiřování open-source kódu. Dále bude proveden zátěžový test v robotické laboratoři, kdy bude měřen vliv počtu připojených robotů a rozsahu síťového toku na odezvu a celkovou paměťovou a výpočetní zátěž serveru.

Zátěž paměti a procesoru je nutné ještě důkladněji změřit během testů v laboratoři. Zátěž sítě je ovšem minimální v porovnání s komunikací nevyužívající Dispatcher, největší vliv na síť má OpenVPN hlavička, která je pro každý packet ve standardním případě 69 bytů. Zpoždění způsobené Dispatcherem při směrování dat je u tří robotů a dvou klientů (maximální dosud testovaný počet) zanedbatelné (¡1ms). Vliv více připojených klientů a robotů je stále nutné změřit.

6. Závěr

V článku 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 rozšířitelnost 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. V budoucnu, pokud bude žádost schválena, bude projekt umístěn i na web stránkách projektu ROS (wiki.ros.org). 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.

Poděkování

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

Literatura

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

[2] *The Robot Management System*. ROS Wiki [online]. [cit. 2016-04-8] Dostupné z: http://wiki.ros/rms.

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

[4] *The Standard ROS JavaScript Library*. ROS Wiki [online]. [cit. 2016-04-8] Dostupné z: http://wiki.ros/roslibjs.

[5] *Rosbridge*. ROS Wiki [online]. [cit. 2016-04-8] Dostupné z: http://wiki.ros/rosbridge_suite.

[6] George Reese. *Cloud application architecture*. O'Reilly Media, Inc., 2009. ISBN: 9780596555481.

[7] Constantinos Dovrolis a Hao Jihang. *Passive estimation of TCP round-trip times*. ACM SIGCOMM Computer Communication Review, 2002. 32(3) 75-88. DOI: 10.1145/571697.571725. ISSN: 01464833.