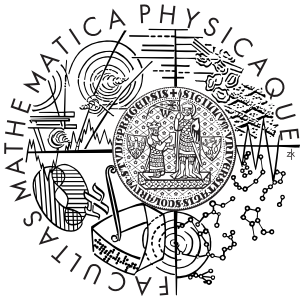
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Ondrej Tichý

Sběr, vizualizace a vyhodnocení dat o provozu metropolitní sítě

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. David Kolovratník
Studijní program: Informatika (ISPS)

2010

Ďakujem pánovi Mgr. Davidovi Kolovratníkovi za odborné vedenie mojej práce, za čas a odporúčania, ktoré mi pri riešení práce venoval.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím uvedenej literatúry. Súhlasím so zapožičiavaním práce.

V Prahe dňa 6.8.2010 Ondrej Tichý

Obsah

1 Úvod **6**

2 Návrh riešenia **8**
2.1 Topológia metropolitných sietí 8
2.2 Agent-klient-server model 9
2.3 Platforma a operačný systém 10

3 Získavanie dát **12**
3.1 Konfigurácia 12
3.2 Modulárny systém 13
3.3 Návrh programu 14
3.4 Odosielanie a dočasné ukladanie dát 16
3.5 Meranie veličín 18
3.5.1 libtraffic.so 18
3.5.2 libcpuusage.so 18
3.5.3 libdiskusage.so 19
3.5.4 libnetworkmap.so 19

4 Ukladanie a poskytovanie dát **22**
4.1 Modulárny systém a dizajn programu 22
4.2 Konfigurácia 23
4.3 Odosielanie a príjem dát 24
4.4 Vytvorenie mapy siete 24
4.4.1 Popis algoritmu 25
4.4.2 Príklad fungovania algoritmu 27
4.5 Ukladanie dát na serveri 30
4.6 Utilita viewgenerator, vytváranie redundancií 31
4.7 Diagnostikovanie problémov v sieti 31

5 Klient **33**
5.1 Qt Framework 33
5.2 Konfigurácia 34
5.3 Zobrazenie mapy siete 34
5.4 Práca so sieťou a prepojenie s modulmi 35
5.5 Zobrazenie grafov 37

6 Porovnanie monitorovacích systémov **38**
6.1 Nagios 38
6.2 MRTG 39
6.3 Zabbix 40

7 Záver **42**

Literatura **43**

A Obsah priloženého CD **45**

B Krátka užívateľská príručka **46**
B.1 Kompilácia 46
B.2 Spustenie a konfigurácia - agent 47
B.3 Spustenie a konfigurácia - server 48
B.4 Spustenie klienta 49

Názov práce: Sběr, vizualizace a vyhodnocení dat o provozu metropolitní sítě
Autor: Ondrej Tichý
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedúci bakalárskej práce: Mgr. David Kolovratník
e-mail vedúceho: David.Kolovratnik@mff.cuni.cz

Abstrakt: V predloženej bakalárskej práci nadväzujem na ročníkový projekt s názvom 'Prevoz smerovača'. Cieľom ročníkového projektu je vytvorenie monitorovacieho systému metropolitnej siete postavenej prevažne na bezdrôtových technológiach. Monitorovací systém má zbierať dáta, ktoré pomôžu získať administrátorovi siete prehľad o stave a predchádzať problémom v sieti vedúcim k výpadkom spojenia. Práca obsahuje podrobné zhodnotenie, ako sa podarilo dosiahnuť zadané ciele. Súčasne obsahuje rutiny na automatické vyhodnotenie zbieraných dát a porovnanie ročníkového projektu s už existujúcimi implementáciami.

Kľúčové slová: Počítačové siete, Wi-fi, Metropolitné siete

Title: Gathering, visualization and evaluation of data about metropolitan computer network
Author: Ondrej Tichý
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Kolovratník David
Supervisor's e-mail address: David.Kolovratnik@mff.cuni.cz

Abstract: In the present thesis I am extending my work on the software project called 'Running a router'. The aim of the software project is to design and create the monitoring system of the metropolitan network based on wireless technologies. The monitoring system should gather, store and present data which would help the administrator get the overvall view of the network and prevent network failures which could lead to the loss of the connectivity. The thesis evaluates the ways how were the given tasks completed. It extends the software project by routines which can automatically evaluate incoming data and inform the administrator and compares the semesteral project to the existing implementations.

Keywords: Computer networks, Wi-fi, Metropolitan network

Kapitola 1

Úvod

Prirodzená potreba ľudí komunikovať vyústila k rozvoju prenosu informácií aj v počítačovom svete. Skomercionalizovaním internetu v 80-tych rokoch nastala prudká penetrácia celosvetovej siete počítačov medzi obyčajných ľudí, čím sa nenávratne zmenil spôsob, akým boli využívané počítače. Prestali byť iba nástrojom na riešenie výpočetných úloh vo firmách, ale stali sa prostriedkom komunikácie, skracovania vzdialeností a v neposlednom rade zábavy.

Podobný trend rozmachu internetových poskytovateľom pozorujeme v 90-tych rokoch aj na Slovensku a Čechách. Ako limitúci faktor pripojenia k internetu sa ukázalo riešenie poslednej míle, teda pripojenia užívateľa k poskytovateľovi. V prvopočiatkoch rozvoja internetu sa ako pripojenie poslednej míle používali už existujúce telefónne kabeláže. Telefónne kabeláže boli dovedené do mnohých domácností a pridaním vhodnej technológie, mohli byť použité aj ako médium pre prenos digitálnych informácií. Položenú telefonnu kabeláž však vlastnili národní telekominukační operátori, čo vyústilo k vytvoreniu monopolu na poskytovanie internetu, potlačeniu trhovej súťaže, prirodzenej regulácie ceny za pripojenie a tým aj k nespokojnosti užívateľov internetu.

Rozvoj a zlacňovanie sieťových technológii a dlhodobá neschopnosť zo strany štátu ovplyvniť reguláciu cien podnietilo užívateľov k hľadaniu alternatívy pre pripojenie k internetu. Spočiatku vznikajú najmä v miestach s vyššou hustotou obyvateľstva komunitné ale aj komerčné siete pokrývajúce jeden urbanistický celok. Takéto siete nazývame metropolitnými sieťami. V súčas-

nosti nachádzame metropolitné siete čoraz častejšie aj na dedinách.

Metropolitné siete dosahujú veľkosti od niekoľkých desiatok až po stovky používateľov. Sú postavené najmä na rádiových bezdrôtových techonológiach IEEE 802.11 a/b/g, alebo pripojením pomocou optického spojenia, napríklad pomocou zariadenia Ronja[8]. Siete takýchto rozmerov sú náročné na administráciu a pomerne náchylné k výpadkom spojenia.

Cieľom tejto práce je navrhnúť spôsob, akým bude metropolitná sieť monitorovaná, popísať a implementovať algoritmy vedúce k získavaniu, ukladaniu a zobrazovaniu veličín sieti. Dôležitým je tiež navrhnúť algoritmus, ktorý automaticky upozorní administrátora na výpadky siete. Práca taktiež porovnáva implementáciu s už existujúcimi známymi programami, ktoré riešia podobnú problematiku.

Druhá kapitola práce opisuje topológiu metropolitných sietí v súčasnosti a zariadenia, ktoré sa v takýchto sietiach používajú. Kapitola pokračuje návrhom programu, ktorý má monitorovať bezdrôtovú sieť. Poslednou časťou kapitoly je súhrn požiadavkov jednotlivých častí programu.

Tretia kapitola je programátorskou dokumentáciou k programu agent. Kapitola obsahuje návrh spôsobu konfigurácie klienta a popis implementácie, návrh architektúry klienta pre monitorovanie jednotlivých veličín. Obsahuje popis modulárneho systému, spôsob akým sú dáta odosielané na server a návrh dočasného ukladania dát v agentovi.

V štvrtej kapitole je popísaný spôsob implementácie programu s názvom server. Podobne ako v agentovi je v krátkosti opísaný spôsob jeho konfigurácie a práca so sieťou. Podrobne je uvedený algoritmus na vytvárenie mapy siete. Nasleduje opis spôsobu ukladania dát na serveri. K programu je prikompilovaná aj utilita na vytváranie vytváranie náhľadov a záloh dát.

Piata kapitola obsahuje krátky opis použitého Framework-u Qt a prehľad najzaujímavejších techník a algoritmov použitých v programe klient.

Šiesta kapitola porovnáva implementáciu s programami podobného zamerania a to Nagios, Zabbix a MRTG. Kapitola obsahuje aj stručnú charakteristiku zmienených programov.

Záver hodnotí dosiahnuté ciele, poukazuje na nedostatky a možné vylepšenia programu.

Kapitola 2

Návrh riešenia

2.1 Topológia metropolitných sietí

Štruktúra metropolitných sietí v čase návrhu programu pozostávala predovšetkým z navzájom pospájaných prípojných bodov, ktoré plnia funkciu smerovačov v sieti. K týmto prípojným bodom sa pripájajú užívatelia metropolitnej siete. Pripojenie je zabezpečené metalickým spojením, pokiaľ sa užívatelia nachádzajú v mieste prípojného bodu. V opačnom prípade sa spojenie nadviaže pomocou bezdrôtovej technológie.

Funkciu smerovania sieťových packetov v prípojnom bode zabezpečuje tzv. prípojný počítač, často označovaný aj ako AP (Access Point). Z dôvodu ušetrenia prostriedkov, či už finančných, alebo energetických, sa ako prípojný počítač najčastejšie používa tzv. krabičkový počítač (embedded system[3]) s osadenými viacerými sieťovými kartami, prípadne starší nepoužívaný počítač. Sieťové karty v prípojnom počítači zabezpečujú premostenie a prepojenie jednotlivých častí siete.

K limitom krabičkového počítača patrí konfigurácia s menej výkonným procesorom a malým úložným priestorom. Často sa dodáva aj úplne bez pevného disku, napríklad s použitím micro SD karty. Počítač sa nedá použiť na výpočetne náročné úlohy, či trvalé uloženie monitorovaných dát. Ako operačný systém prípojného počítaču sa v dobe písania práce vo väčšine prípadov používal systém GNU/Linux [1], iba vo výnimočných prípadoch klon operačného

[1] Linux 92%, FreeBSD 8%, Windows 0%. Údaje pochádzajú zo siete autora práce.

systému typu BSD a skoro vôbec sa nestretávame s použitím operačného systému Windows.

Sieť postavenená s navzájom poprepájaných prípojných bodov vo väčšine prípadov vedie k počítaču, ktorý zabezpečuje pripojenie do internetu nazývanému aj Internet gateway. Obsahuje zväčša väčší pevný disk, ktorý často slúži aj ako server pre elektronickú poštu (SMTP), databázu (MYSQL), úložisko dát (FTP), internetové prezentácie (HTTP), či správu užívateľov autorizovaných pre pripojenie do siete.

2.2 Agent-klient-server model

Z typickej topológie siete vyplýva, že pre fungovanie siete je nevyhnutný bezproblémový beh prípojných bodov. Prípojné body však podliehajú mnohým vonkajším vplyvom, ktoré negatívnym spôsobom ovplyvňujú ich funkčnosť.

Väčšina komunitných metropolitných sietí je postavená na bezdrôtových technológiach vo voľných licenčných pásmach 2,4GHz a 5GHz. Prenos vo voľných licenčných pásmach je regulovaný iba v obmedzenej miere a tak dochádza k častému rušeniu signálu, čo sa odzrkadlí v horšej kvalite linky, alebo až v celkovej nedostupnosti siete.

Nezanedbateľnými vplyvmi na stabilitu siete patria aj poruchy prípojného počítaču ako preťaženie procesora, málo miesta na disku, či nadmerné vyťaženie siete užívateľmi.

Z uvedených príčin sa javí vhodné monitorovať funkcie prípojného bodu programom umiestneným priamo na počítači prípojného bodu. Takýto program bude zbierať dáta o prípojnom bode. Z limitujúcich faktorov konfigurácie prípojného počítaču však plynú pre program obmedzenia ako nemožnosť predpokladať dostatok miesta na disku. Preto dáta pre trvalé uloženie odošleme na počítač bez limitujúcich faktorov.

Program, ktorý získava dáta v prípojných bodoch, nazveme **agent**. Samotné spojenie, však nemusí byť vždy dostupné a mohlo by tak dôjsť k strateniu získaných dát. Preto je nutné monitorované dáta v prípade výpadku spojenia dočasne uložiť a počkať s odoslaním až do momentu, kedy spojenie bude znova dostupné.

V centrálnom uzle, resp. na vhodnom počítači v sieti, vytvoríme program

s názvom **monitorovací server**, ktorý sa stará o ukladanie a poskytovanie dát pre agentov a klientov. Súčasne vhodným sposobom upravuje dáta, umožňuje vytváranie náhladov v rôznych rozlíšeniach a v prípade výpadku spojenia o nich informuje klientov.

Posledným programom je klient. **Vizualizačný klient** je program, ktorý zobrazuje uložené dáta a užívateľovi poskytne základný prehľad o štruktúre siete.

Klient sa môže k monitorovaciemu serveru pripojiť z ľubovoľného miesta na internete, v prípade ak administrátor zabezpečí spojenie s počítačom, ktorý bude poskytovať funkciu brány prístupu k internetu.

Dôležitým aspektom pri návrhu a implementácii agenta a serveru je dôraz na to, aby oni sami nevyťažovali procesor, nakoľko by tak bránili užívaniu siete.

2.3 Platforma a operačný systém

Agent získava merané hodnoty pomocou modulov, ktoré využívajú systémovo závislé súbory a volania operačného systému. Nakoľko drvivá väčšina prípojných bodov obsahuje nainštalovaný operačný systém GNU/Linux je aj agent závislý na tomto operačnom systéme. Program bol testovaný na jadre operačného systému verzie 2.6. Závislosť na distribúcii nie je, program bol však testovaný na distribúcii Linuxu Ubuntu 9.04. Jadro agenta by malo byť spustiteľné na akomkoľvek Unix-like operačnom systéme.

Medzi použitými knižnicami je agent závislý na knižnici liblcfg [5], ktorú používa na načítanie konfigurácie. Táto knižnica je veľmi malá, preto som sa rozhodol ju prikompilovať k zdrojovému kódu aplikácie pomocou zhrnutia jednotlivých hlavičkových súborov do jedného. Agent teda nie je závislý od nainštalovania knižnice liblcfg.

Pre beh vlákien som použil knižnicu libpthreads. Knižnica implementuje vlákna vyhovujúce norme POSIX. V nasledujúcich bodoch zhrniem jednotlivé systémové požiadavky pre program agent.

- Operačný systém - GNU/Linux, Kernel 2.6.XX.
- Procesor - i386, amd64 (little endian byte order)

- Programovací jazyk - C
- Použité knižnice a utility - liblcfg, libpthread, sed, df, cat

Server implementuje podobne ako agent moduly, tie však nie sú závislé na použitom operačnom systéme. Pre správne fungovanie servera, je potrebný Unix-like operačný systém a procesor typu little endian. Program bol testovaný na operačnom systéme Debian 9.04, preto je uvedený ako podporovaný operačný systém. Pre spustenie serveru je teda potrebné nasledovné:

- Operačný systém - GNU/Linux, Kernel 2.6.XX
- Procesor - i386, amd64 (little endian byte order)
- Programovací jazyk - C
- Použité knižnice a utility - liblcfg, libpthread, mail, wget.

Klient zobrazuje dáta graficky, preto pri implementácii bolo nevyhnutné použitie grafickej knižnice. Použitie Qt [12] garantovalo ľahké portovanie klienta na ostatné operačné systémy a súčasne inklinovalo k použitiu jazyka C++, nakoľko väčšina dokumentácie a príkladov je napísaných v tomto jazyku.

Medzi ostatné závislosti patrí sqlite3 [11], ktoré slúži k implementovaniu databázy. Program je spustiteľný pod operačnými systémami ktorých podpora je pododrobne rozobraná v [13].

Na vykreslovanie grafov som sa rozhodol použiť knižnicu Qwt[15]. Knižnica poskytuje základ pre statické kreslenie grafov a rozširuje tak možnosti frameworku. Pre jej spustenie je nevyhnutné mať správne nastavenú systémovú premennú LD_LIBRARY_PATH.

- Operačný systém - GNU/Linux, Windows XP
- Programovací jazyk - C++
- Použité knižnice - Qt, Qwt, sqllite

Kapitola 3

Získavanie dát

3.1 Konfigurácia

Konfigurácia agenta prebieha pomocou konfiguračného súboru. Konfiguračný súbor sa načíta pri použití prepínaču -c pri štarte. Nešpecifikovanie konfiguračného súboru má za následok nespustenie programu, nakoľko niektoré parametere sú nevyhnutné pre beh agenta. Je nutné nastaviť minimálne meno agenta, ktoré slúži ako identifikátor, IPv4 resp. IPv6 adresu monitorovacieho serveru a súčasne port, ktorý identifikuje server ako bežiacu službu.

Ostatné parametre ovplyvnia funkčnosť jednotlivých modulov. Parameter, ktorý musí byť špecifikovaný na to, aby sa modul načítal pri behu programu je parameter *_time*. Čas určuje, ako často má byť volaná funkcia modulu na zmeranie žiadanej veličiny. Nepovinným parametrom je parameter *run_module_in_thread*, ktorý ovplyvňuje sposôb, akým je meraná daná veličina.

Načítavanie konfigurácie prebieha pomocou knižnice liblcfg [5], napísanej v jazyku C99. Pre túto knižnicu som sa rozhodol z dôvodu jej zanedbateľnej veľkosti, prenositelnosti a jednoduchosti použitia. Medzi ostatnými knižnicami s podobnými, častokrát lepšími, vlastnosťami však bola patrná závislosť na jazyku C++, prípadne závislosť na inom operačnom systéme.

Dáta z konfiguračného súboru sú potrebné naprieč celým programom. Z tohto hľadiska sa javí ako neefektívne rozširovať sieť volaní funkcií o parametre reprezentujúce globálne vlastnosti programu.

V jazyku C++ existuje pomerne jednoduché a efektívne riešenie. A to instanciovať globálnu triedu obsahujúcu metódy pre prístup ku konfiguračným parametrom. Konštruktor tejto triedy sa postará o skontrolovanie formátu konfiguračného súboru a o načítanie hodnôt kľúčov do privátnych premenných triedy. Z vlastností tejto triedy je patrné, že bude existovať iba vo forme jednej instancie reprezentujúcej konfiguračný súbor. Trieda s jednou instanciou sa nazýva singleton.

Jazyk C nie je objektovo orientovaný a taktiež neobsahuje konštrukcie môžné vytvoriť singleton. Pokúsil som sa ho však simulovať štruktúrou *_config*, ktorá obsahuje položky reprezentujúce jendotlivé konfiguračné nastavenia. Táto štruktúra má jediný globálny výskyt. Nakoľko používanie globálnych premenných je považované za zlý programátorský zvyk, pre prístup k tejto štruktúre používam v programe funkciu *getConfig*, ktorá vráti ukazateľ na túto štruktúru. Funkcia *read_configuration* volaná na začiatku programu sa stará podobne ako konštuktor u singleton-u o načítanie konfiguračného súboru a spracovanie parametrov. Táto funkcia je teda volaná iba raz a to pri štarte programu.

Nevýhodou jazyka C je nemožnosť zapúzdrenia parameterov modulu, podobne ako sú zapúzdrené privátne premenné tried v jazyku C++. Prístupové funkcie k jednotlivým položkam štruktúry *_config* som však neimplementoval z dôvodu, že takáto sada funkcí, ktorá by nebola priamo prístupná v konfiguračnej štruktúre, by zneprehľadnila samotný program.

3.2 Modulárny systém

O získavanie a monitorovanie veličín sa starajú entity programu nazvané moduly. Modul je dynamicky načítateľná knižnica (dynamicly loaded library), ktorá obsahuje funkcie ovplyvňujúce beh programu.

V konfiguračnom súbore sa špecifikuje parameter *modules_directory*, ktorý reprezentuje priečinok, kde sa nachádzajú jednotlivé moduly. Moduly obsahujú vopred známe rozhranie definovanom v súbore agent_module_interface.h. Modul je teda povinný implemetovať funkciu pre získanie dát *get_data*, pričom dáta vráti v štruktúre zaobalujúcu všetky dáta z modulov.

K získaným dátam modul pridá čas, kedy boli dáta odmerané. Priestor pre dáta alokuje na halde, pričom ukazateľ na toto miesto, dĺžku dátovej

položky a ich počet uloží do štruktúry *data_return*. O odoslanie štruktúry sa starajú vyššie vrstvy programu.

Systém dynamicky načítateľných knižníc dovoľuje užívateľovi špecifikovať, ktoré dáta chce monitorovať tak, že knižnicu pre danú funkcionalitu nakopíruje do adresára *modules_directory*. K výhodám tohto systému patrí jeho ľahká konfigurovateľnosť, šetrenie miesta na disku tým, že je skutočne prítomná iba tá časť programu, ktorá sa stará o monitorovanie žiadaných veličín. Podobný systém modulov obsahuje napríklad známy HTTP server Apache.

Implementoval som moduly

- libcpuusage.so, slúži na meranie vyťaženia procesoru,
- libdiskusage.so, meria voľné miesto na disku,
- libtraffic.so, meria využitie sieťových kariet,
- a libnetworkmap.so získava mapu siete.

Pridanie nového ďalšieho modulu predstavuje naimplementovanie funkcie *get_data*, podľa špecifikácie. Následné nakopírovanie modulu do adresára, ktorý bude agent prehľadávať a špecifikovanie času, ako často sa bude nová veličina merať v konfiguračnom súbore tak, že sa odstráni prípona .so a doplní sa koncovka *_time*.

3.3 Návrh programu

Program po spustení načíta ukazatele na jednotlivé meracie funkcie. Meracia funkcia by mala v pravideľných intervaloch odmerať žiadanú veličinu a odovzdať ju programu, ktorý odmerané veličiny odošle na monitorovací server.

Triviálnym riešením je napísať program ako nekonečný cyklus, ktorý sa uspí na základnú dobu. Program by si pamätal, celkový čas, ktorý funkcia prespala a v prípade, že by celkový čas prekročil celočíselný násobok intervalu, kedy by mali byť spustené jednotlivé moduly, zavolal by program funkciu

na získanie dát. Súčasne pokiaľ by program v priebehu odmeral nejaké dáta, zavolal by obslužnú rutinu na ich odoslanie.

Problémom v zadanom riešení je odosielanie dát. Pripojenie sa na monitorovací server, rovnako ako všetky obslužné rutiny pre prácu so sieťou majú blokujúci charakter. Keďže meranie aj odosielanie prebieha v jednom vlákne, dochádzalo by k meškaniu v meraní dát. Pri obzvlášt krátkych intervaloch (cca. do 5 sekúnd) by mohlo dôjsť aj k výpadkom merania.

Blokovaniu dát pri odosielaní by sa dalo zabrániť použitím systémového volania *poll*. Systémové volanie poll otestuje, či je zadané pole file descriptor-ov[1] pripravné na čítanie. Volania *epoll* a *select* pracujú obdobne, líšia sa iba v efektivite resp. v rozhraní. Agent by pred samotným (blokujúcim) pripojením, alebo prečítaním správy skontroloval, či už má file descriptor socket-u pripravený na písanie, alebo zápis. V prípade, že nie, pokračoval by vo svojej práci. Použite tejto konštrukcie by umožnilo spustiť program v jednom vlákne bez nutnosti čakania na pripojenie agenta, či prijatie potvrdenia uloženia dát na pomalej linke.

Okrem zablokovania pri práci so sieťou môže dôjsť k čakaniu na jednotlivé moduly počas ich merania. Pri získavaní informácií o štruktúre siete dochádza k spusteniu utility traceroute, ktorá zistí cestu od agenta k monitorovaciemu serveru. Meranie pomocou tejto utility môže v závislosti od rýchlosti siete a odozvy jednotlivých prípojných bodov trvať rádovo desiatky sekúnd. Čakanie v tomto rozsahu znamená typicky výpadok niekoľkých cyklov merania ostatných veličín.

Zablokovanie pri meraní veličín môžeme vyriešiť pomocou volaní *epoll*, avšak rozhranie pre prístup k týmto funkcíam by sa značne zkomplikovalo, nakoľko by si opakované volanie funkcie *get_data* na module vyžadovalo zapamätanie, kde v predchádzajúcom volaní skončilo.

Z dôvodu nepraktickosti tohto riešenia sme sa rozhodoli oddeliť meranie veličín, ktorých meranie prebieha dlho do samostatného vlákna. Rovnako z dôvodu čistoty návrhu sme oddelili aj odosielanie dát do samostatného vlákna s tým, že odosielanie prebieha v užívateľom stanovených pravidelných intervaloch.

Modul sa skonfiguruje pre beh vlákna nastavením parametru *run_in_thread*

[1]Pod pojmom file descriptor chápe číslo, pod ktorým si pamätá operačný systém ukazateľ na súbor, zariadenie, spojenie.[4]

na kladnú hodnotu.

K implentovaniu behu vlákien bola použitá knižnica pthreads. Knižnica vyhovuje norme POSIX. Vlákno beží v nekonečnom cykle, pričom sa sa uspí na interval, počas ktorého spustí rutinu modulu na meranie dát, resp. rutinu na odosielanie dát.

Prostredníkom na predávanie dát medzi vláknami a hlavným programom je spojový zoznam. Vstup do spojového zoznamu je zdielaný, preto je nevyhonutné vlastniť zámok, aby nedošlo k porušeniu konzistencie spojového zoznamu. Samotný zdrojový kód pochádza z jadra operačného systému Linux verzie 2.6. V implementácii sme rozšírili spojový zoznam o viaceré funkcie a podporu pre beh prístup viacerých vlákien pomocou mutex-u[2].

3.4 Odosielanie a dočasné ukladanie dát

V predchádzajúcej kapitole bol objasnený spôsob, akým sa dáta získavajú a predávajú vláknu, ktoré odosiela dáta. Vlákno, ktorého úlohou je odosielať dáta, po zavolaní odosielacej rutiny najprv skopíruje spojový zoznam s dátami, ktoré má odoslať do nového spojového zoznamu, aby nedošlo k narušeniu dát.

Následne zavolá funkciu *initialize_server_network*, ktorou sa pokúsi skontaktovať so serverom. Agent vytvára spojované spojenie TCP na IP adresu a port zadaný v konfiguračnom súbore. Spojované spojenie bolo použité z dôvodu, že agent sa stará predovšetkým o posielanie konzistetných dát, pričom výpadok ktorejkoľvek časti dát by znamenalo porušenie ich konzistencie. Preto by bolo nutné naimplementovať vlastné rutiny, ktoré by iba simulovali chod spojovaného spojenia.

V prípade, že sa agentovi nepodarí úspešne inicializovať spojenie, pokúsi sa dáta dočasne uložiť na pevnom disku. O ukladanie sa stará funkcia *cache_data_return*. Dáta sú ukladané do súboru špecifikovanom v konfiguračnom súbore parametrom *cache_file*. Nakoľko k tomuto súboru pristupuje jediné vlákno, nie je potrebné získať k prístupu do súboru zámok.

V súbore sú dáta uložené v tom poradí, v akom ich jednotlivé moduly získali. Je tak dodržaná konzistencia uloženia údajov. Platí, že dáta z jedného mo-

[2]Synchronizačný prvok pre prístup k zdielaným dátam [9]

dulu sú uložené podľa času vzostupne od začiatku do koncu súboru. Môžu sa prekrývať rôzne časové známky rôznych modulov, no pri odosielani dát je dodržané toto pravidlo.

Po úspešnom pripojení a autentifikácii je agent pripravený na odosielanie dát k jednotlivým modulom. Dáta najprv zabalí do formátu vhodnom pre odoslanie po sieti pomocou funkcie *pack_module_data*. Nakoľko drvivá väčšina počítačov v menších metropolitných sietiach je typu *little_endian*[1], neboli naimplementovalné rutiny, ktoré by brali v úvahu iné poradie byte-ov, ako poradie byte-ov na počítačoch tohto typu.

Funkcia pack_module_data vloží dáta v nasledovnom poradí:

- meno modulu,
- počet dát,
- dĺžka dát,
- časová známka
- dáta získané modulom.

Dáta sa odosielajú v tomto poradí. Najprv sa agent pokúsi odoslať súbor s dátami, ktoré boli dočasne uložené. Dáta číta v cykle, pomocou funckcie *read_module_data*. V prípade, že nastane porucha počas spojenia, zapamätá si pozíciu v súbore, pokiaľ boli dáta odoslané.

V prípade poruchy sa vytvorí dočasný súbor s príponou .tmp, kam sa nakopírujú neodoslané dáta od zapamätanej pozície. Týmto sa súbor po dokončení kopírovania prepíše na povodný a nastaví sa príznak poruchy odosielania.

V prípade úspešného odoslania sa súbor vymaže a pokračuje sa v odosielaní aktuálnych dát čakajúcich vo fronte. V prípade, že nastane chyba spojenia v tomto momente, sú nové dáta prekopírované na koniec súboru.

[1]Počítače tohto druhu ukladajú dáta byte s najvyššou hodnotou vpravo.

3.5 Meranie veličín

V nasledujúcej časti práce je popísané, akým sposobom získavajú moduly na strane agenta dáta a taktiež uvedieme dôvody, prečo je dôležité monitorovať práve tieto veličiny na strane prípojných bodov.

3.5.1 libtraffic.so

Monitorovanie sieťového trafficu prebieha periodickým čítaním hodnôt sieťových adaptérov *rx_bytes* a *tx_bytes* zo súboru */proc/dev/net*. Súbor */proc/dev/net* obsahuje zoznam sieťových rozhraní, ku ktorým nasleduje výčet parametrov jednotlivých hodnôt.

Súbor sa číta postupne, pričom dáta sú ukladané do štruktúry *struct traffic_data* pomocou regulárnych výrazov v knižnici regex.h. Knižnica regex.h je súčasťou štandartu Unix v [17].

Monitorovanie sieťovej prevádzky je dôležitým aspektom pri skúmaní vyťaženia jednotlivých liniek v sieti. Pomáha administrátorovi zistiť, ktoré časti siete užívatelia využívajú najviac.

3.5.2 libcpuusage.so

Monitorovanie využitia procesoru prebieha čítaním súboru */proc/stat*. Súbor */proc/stat* obsahuje zoznam procesorov inštalovaných v prípojnom počítači. Procesor s názvom cpu spokytuje hodnoty zo všetkých procesorov súčasne. Monitorujú sa hodnoty user, nice, total, idle.

Monitorovanie využitia systémových prostriedkov pomáha zistiť, či prípojný počítač stačí na úkony, ktoré má zadané. Rovnako vysoké hodnoty využitia môžu značiť infiltrácie malware[3] v sieti.

[3]Malware je škodlivý software, ktorý infiltruje počítače bez vedomia administrátora.

3.5.3 libdiskusage.so

Knižnica monitoruje využitie pevného disku. Nakoľko počítač môže obsahovať viacero pevných diskov, monitoruje sa počet zapísaných byte-ov na všetkých partíciach a počet dostupných bytov na partíciach.

Dáta sa získavajú pomocou utilít *df* a *sed*. Program vytvorí nový proces pomocou funkcie *popen*, kde zavolá utilitu *df*, ktorá vypíše zoznam jednotlivých pripojených partícií. Zoznam partícií sa cez pipe predá filtru *sed*. Filter sed zmení výstup utility df na zoznam zapísaných a dostupných bytov oddelených medzerou. Príklad použitia uvádzam vo vloženom kóde 3.1.

Listing 3.1: Príklad použitia filtru sed

```
:~\$ df
Filesystem  1K-blocks Used       Available Use% Mounted on
/dev/sda2   22105792  14644796   6338056   70% /
none        971384    336        971048    1% /dev
none        975892    752        975140    1% /dev/shm
none        975892    292        975600    1% /var/run
none        75892    0          975892    0% /var/lock
none        75892    0          975892    0% /lib/init/rw
/dev/sda4   100582964  82881748  17701216  83% /media/storage

:~\$ df | \
sed -n '
s/^\(\/dev[^ ]*\) *\([0-9]*\) *\([0-9]*\) *\([0-9]*\).*\$/\3 \4/gp;
'
14644796 6338056
82881748 17701216
```

Výstup funkcie *popen* je následne otovorený ako file descriptor v programe, ktorý jednoducho prečíta funkcia scanf. Jednotlivé položky sú sčítané a zabalené funkciou *pack_disk_usage* do formátu vhodného pre odoslanie po sieti.

Monitorovanie diskového priestoru je obzvlášť dôležité pri prípojných bodoch s malým diskovým miestom, resp. prípojných bodoch, ktoré slúžia užívateľom ako diskové úložisko dát (ftp).

3.5.4 libnetworkmap.so

Modul network map sa stará o získanie mapy siete. Získanie dát k skonštruovaniu mapy siete zo strany agenta obnáša odoslať informáciu o mene

agenta, jednotlivých sieťových rozhraniach a k nim prislúchajúcim sieťovým adresám IP. Druhou časťou dát je zoznam IP adries na ceste od agenta k monitorovaciemu serveru.

Meno agenta získa z konfiguračného súboru. Následne zistí počet sieťových rozhraní zo súboru /proc/net/dev s použitím funkcie ndev. K sieťovým rozhraniam naalokuje pole štruktúr *iface_settings*, do ktorých zabalí jednotlivé údaje.

Získanie sieťovej adresy je závislé na operačnom systéme GNU/Linux. Adresu získa funkcia *get_ipv4* resp.*get_ipv6*. Funkcia dostane ako parameter meno rozhrania a vráti sieťovú adresu. Funkcia otvorí socket pomocou volania *socket(AF_INET, SOCK_DGRAM, 0);* a následne na otvorený file descriptor zavolá funkciu *ioctl*, ktorá do parameterov socketu vpíše sieťovú adresu. V prípade, že návratovou hodnotou z funkcie bude -1, sieťová karta nemá inicializovanú sieťovú adresu a v zozname sieťových adries sa neuvedie.

Zo zoznamu sa taktiež vynechá loopback rozhranie s názvom loX. Toto rozhranie sa vyskytuje na každom počítači s rovnakou sieťovou adresou, preto identifikovanie agenta na základe takejto sieťovej adresy nie je jednoznačné a teda pre potreby získania mapy siete, ju nemá zmysel odosielať.

Druhou časťou dát je zoznam IP adries na na ceste od agenta k monitorovaciemu serveru. Zoznam adries sa získa podobným spôsobom ako zoznam sieťových rozhraní.

Ako prvý krok skonštruujeme príkaz, ktorý máme spustiť. Použijeme utilitu traceroute, ktorej ako parameter dáme meno monitorovacieho servera.

Utilita traceroute odošle IP packety s cieľovou adresou monitorovacieho serveru na sieti. V packete je uvedená hodnota TTL[4]. Úlohou každého smerovača po sieti je túto hodnotu znížiť o jedna a následne packet odoslať ďalšiemu smerovaču po ceste. V i-tom kroku hodnota TTL reprezentuje počiatočnú hodnotu TTL zníženú o hodnotu i.

V prípade, že by mal smerovač túto hodnotu znížiť na 0, packet ďalej neposiela, ale namiesto toho informuje odosielatela packetu ICMP správou o tom, že packet zahodil a teda už nikdy nedorazí do cieľa.

Túto vlastnosť využíva utilita k zisteniu jednotlivých IP adries na ceste smerovačov tak, že hodnotu TTL zvyšuje až do momentu, kedy packet nedorazí

[4]Time to live, štandartná hodnota je 128.

na monitorovací server.

Dôvodom na takéto správanie je prevencia zahltenia siete cyklením packetov v prípade chybného nastavenia smerovacích tabuliek smerovačov.

Na výstup z traceroute použijeme filter sed z ktorého zobrazíme iba jednotlivé adresy. Tie naalokujeme do statickej štruktúry, nakoľko počet skokov od monitorovacieho agenta k monitorociemu serveru je obmedzený.

Kapitola 4

Ukladanie a poskytovanie dát

4.1 Modulárny systém a dizajn programu

Monitorovací server rovnako ako agent pracuje pomocou dynamicky načítateľných modulov. Po spustení prehľadá monitovací server dynamicky načítateľné knižnice a nájde v nich funkcie *select_data* a *store_data*, ktoré sa starajú o ukladanie a vybratie dát. Ich rozhranie je definované v súbore server_module_interface.h, ktoré musí každý modul implementovať. Modul na strane servera sa stará o uloženie, spracovanie a získanie dát. Každému modulu na strane agenta, odpovedá modul na strane servera.

Podobne ako v agentovi, aj na strane serveru je nutná potreba paralizovať požiadavky zo strany klienta resp. agenta. Agent resp. klient, nemôže čakať na uloženie ostatných dát.

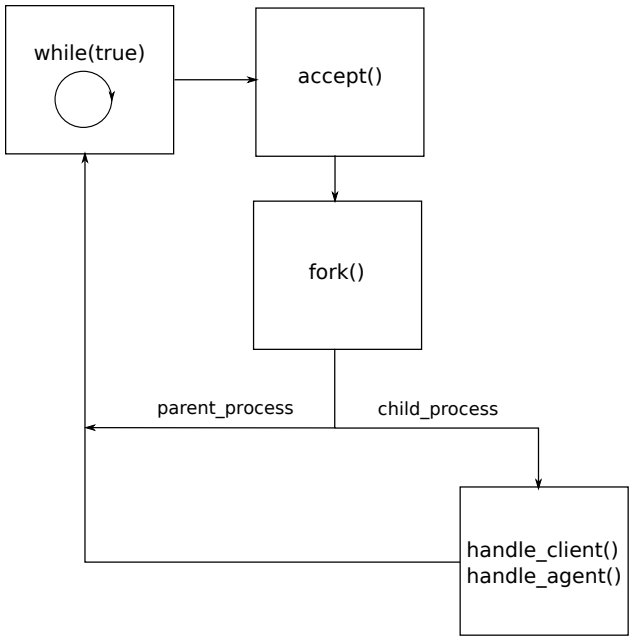
Paralelizácia sa dá realizovať podobne ako v agentovi pomocou testovania poľa file descriptor-ov a ich prípravu na zápis a čítanie dát pomocou funkcí *poll*, *epoll* alebo *select*. Rovnako ako na strane agenta by bol návrh modulárnych funkcií komplikovaný.

V ďalšom texte budem pod pojmom klient chápať ľubovoľný z dvojice programov agent resp. vizualizačný klient.

Po načítaní modulov a konfigurácie beží program v nekonečnom cykle. Po pripojení klienta sa pomocou volania *fork* skopíruje proces aj s premennými do nového synovského procesu, ktorý sa stará o vybavanie požiadaviek agentov

a klientov. Po ukončení vráti otcovskému procesu návratovú hodnotu 0-255, ktorá predstavuje úspech alebo neúspech behu.

Otcovský proces sa v cykle vráti a zablokuje na ďalšom volaní *accept*. Životný cyklus servera môžeme vidieť v diagrame 4.1.



Obr. 4.1: Diagram znázorňujúci životný cyklus serveru.

4.2 Konfigurácia

Konfigurácia servera je obdobná ako u agenta. Server používa knižnicu liblcfg, pričom okrem štandartných funkcií používa aj funkcie lfgx_tree.c, ktoré umožnia využiť okrem základných reťazcov aj vnorené konfiguračné nastavania, zoznamy a mapy.

Konfigurácia je uložená v štruktúre *_config*. Štruktúra je globálnou premennou, ukazateľ na ňu vracia funkcia *getConfig*. Ide teda znova o simulovanie C++ objektu s jednou instanciou - singleton.

Novými konfiguračnými nastaveniami sú prístupové súbory špecifikované konfiguračným nastavením *agent_auth_list* a *client_auth_list*.

4.3 Odosielanie a príjem dát

Odosielanie dát ako aj autentifikácia prebieha podľa jednoduchého protokolu.

V prvom kroku sa server pripojí systémovým volaním na port špecifikovaný v konfiguračnom súbore. Následne počúva na porte až do momentu, kedy sa na ňom objaví nové spojenie.

Po vytvorení TCP spojenia agent odošle informáciu začínajúcu kľúčovým slovom AGENT, klient odošle kľúčové slovo KLIENT. Za nimi nasleduje meno a heslo tak, ako ich majú špecifikované v konfiguračnom súbore.

Po úspešnom odoslaní server sa server pokúsi nájsť konfiguračný súbor a v ňom príslušnú dvojcu meno a heslo. Pokiaľ v ňom úspešne vyhľadá prístupové meno a heslo, informuje o tom práve pripojeného agenta resp. klienta. Pokiaľ konfiguračný súbor nenájde, povolí prístup agenta resp. klienta na server. Autentifikácia zlyhá, ak monitorvací server nájde konfiguračný súbor, ale nenájde v ňom odpovedajúcu dvojcu meno a heslo.

Následne prijíma dáta v cykle a predáva ich jednotlivým modulom. Agentovi po prijatí dát odosiela informáciu o tom, že môže odosielať ďalšie dáta.

Klientovi modul vyberie dáta, vráti ich serveru a server ich odošle späť po inicializovanom spojení.

Spojenie sa končí ukončením spojenia na strane agenta resp. klienta. Server sa o tom dozvie tak, že volanie *recv* vráti hodnotu -1.

4.4 Vytvorenie mapy siete

O vytvorenie mapy siete sa stará modul libnetworkmap.so na serveri. Po pripojení odošle agent informáciu o časti siete.

Formát dát na strane agenta reprezentuje

- zoznam sieťových kariet a
- zoznam jednotivých skokov vzniknutých od agenta, získaných utilitou traceroute smerom k monitorovaciemu serveru.

Sieťové karty sú identifikované menom sieťového adaptéru a sieťovými adresami (IPv4, IPv6). Skoky získané utilitou traceroute sú identifikované sieťovými adresami. Utilita traceroute je spustená s parametrom, ktorý neprekladá DNS mená ale zobrazuje priamo sieťové adresy.

4.4.1 Popis algoritmu

Monitorovací server si po prijatí dát vytvára súbor so zoznamom agentov, ku ktorým si poznačí ich identifikátor, definovaný ako pole znakov o konštatnej dĺžke. Agenta identifikuje funkcia *identify_agent*, ktorá ako parameter dostáva sieťovú adresu a vracia identifikátor. Akonáhle server prijme nové dáta, tento zoznam zaktualizuje a súčasne, ak došlo k zmene dát prijatých od agenta, zavolá funkciu *update_network_map*.

Mapa siete je na serveri uložená ako zoznam susedov jednotlivých prvkov siete. Sused je reprezentovaný štruktúrou _id. Jednotlivé spojenia získané utilitou traceroute sú obojstranné. Graf reprezentovaný zoznamom susedov je teda neorientovaný. Jeho reprezentácia dátovou štruktúrou ho ukladá ako orientovaný. Platí teda, že ak je v zozname susedov vrcholu u hrana e spájajúca vrcholy $< u, v >$, bude v zozname vrcholu v hrana f spájajúca vrcholy $< v, u >$.

Funckia *create_network_map* vytvorí mapu siete zo zoznamu agentov. Funkcia začne prechádzať jednotlivých cesty agentov po dvojciach. Pre každú dvojcu susedných IP adries nájde pomocou funkcie *identify_agent* prislúchajúce meno agenta. V prípade, že meno nenájde, je zaručené, že ho nenájde ani v žiadnom ďalšom kroku, nakoľko súbor s agentami bol vytvorený a aktualizovaný pred vstupom. Preto použije ako dočasný identifikátor agenta jeho IP. Pre úplnosť uvádzam pseudokód 4.1

Listing 4.1: Pseudokód k funkcii create_network_map

```
create_network_map(agent_record list) {
  neighbour_list

  forall agent_data in list {
    forall ip_addr in agent_data.list {
      current_agent = identify_ip(ip_addr);
      left_agent = identify_ip(ip_addr.prev);
      right_agent = identify_ip(ip_addr.next);
```

```
    }

    update_list(neighbour_list, current_agent, left_agent);
    update_list(neighbour_list, current_agent, right_agent);

    update_list(neighbour_list, left_agent, current_agent);
    update_list(neighbour_list, right_agent, current_agent);
  }
}
```

Funckia *update_list* aktualizuje zoznam susedov o novú dvojcu. Funkcia dostane ako parameter zoznam aktuálny zoznam agentov a k nim prislúchajúcich susedov.

V prvom kroku sa pokúsi funkcia prejsť zoznam agentov. Pokiaľ bude súhlasiť identifikátor agenta s novým pridávaným, pridá mu suseda, ktorý dostala ako posledný parameter. Pokiaľ by bol zoznam susedov k prislúchajúcemu agentovi implementovaný ako pole, či spojový zoznam, je nutné, aby funkcia nepridávala agenta s rovnakým identifikátorom dvakrát. Preto pridanie do zoznamu v mojej implementácii (spojovým zoznamom) má zložitosť $O(N)$ napriek tomu, že bežné pridanie prvku do spojového zoznamu má konštatnú časovú zložitosť $O(1)$.

Pokiaľ funkcia nenájde v zozname agentov hľadaný identifikátor, pridá ho ako prvok na koniec zoznamu, inicializuje jeho zoznam susedov a do tohto zoznanu pridá agenta identifikovaného posledným parametrom.

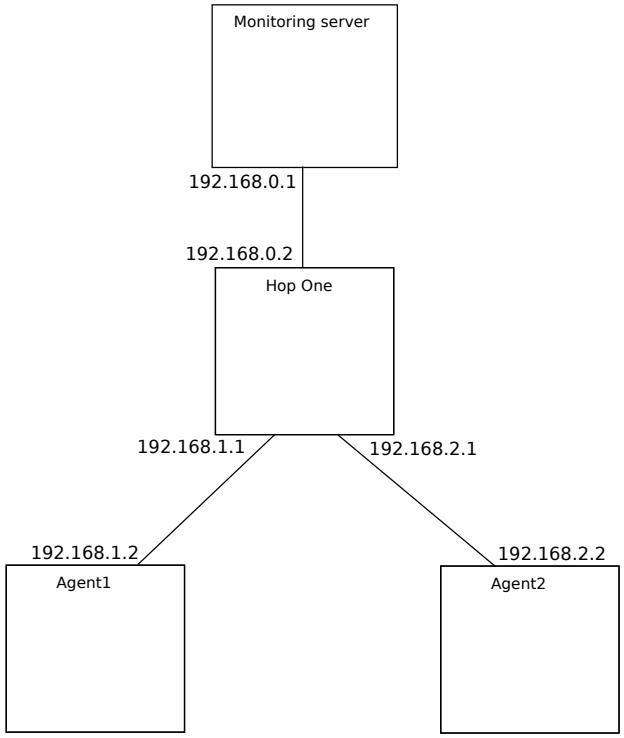
Listing 4.2: Pseudokód k update_list

```
update_list(agent_neighbour_list, current_agent, new_agent) {
  forall agent in agent_neighbour_list {
    if (agent.agent == current_agent) {
      if (new_agent not in agent.neighbours) {
        add(new_agent, agent.neighbours);
    }
  }

  if (current_agent not in neigbour_list) {
    agent_neighbour = init_list(current_agent)
    add(new_agent, agent_neighbour)
    add(agent_neigbour, neighbour_list);
  }
}
```

4.4.2 Príklad fungovania algoritmu

Na nasledujúcom diagrame 4.2 je zakreslená jednoduchá štruktúra siete, na ktorej budem demonštrovať fungovanie algoritmu v reálnych podmienkach. Sieť obsahuje trojcu agentov. Agent1 so sieťovým rozhraním eth0 s nastavenou IP adresou 192.168.1.2. Agent2 so sieťovým rozhraním eth1 a nastavenou IP adresou 192.168.2.2. Agent1 a Agent2 sú priamo spojení so sieťovými rozhraniami agenta Hop s adresami 192.168.1.1 (eth1) pre Agenta1 a 192.168.2.1 (eth2) pre Agenta2. Agent Hop je pripojený sieťovým rozhraním eth0 s IP adresou 192.168.0.2 k monitorovaciemu serveru s jediným sieťovým rozhraním eth0 s nastavenou IP adresou 192.168.0.1.



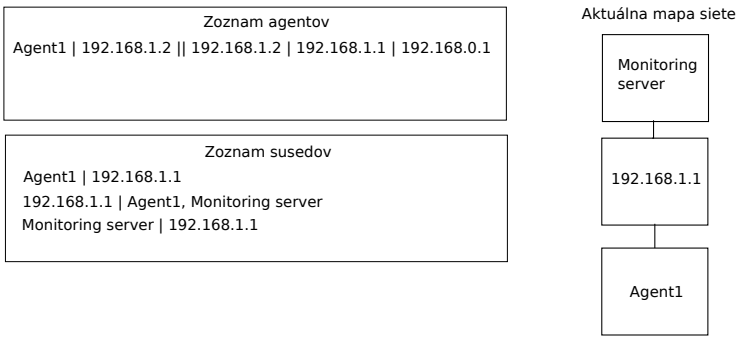
Obr. 4.2: Diagram siete

V prvom kroku odošle informáciu o stave siete Agent1. Agent1 má okrem lo0 jedno aktívne sieťové rozhranie eth0 na ktorom má nastavenú IP adresu 192.168.1.2. Týmto sieťovým rozhraním a utilitou traceroute odoslal informácie na monitorovací server o ceste k monitorovaciemu serveru obsahujúce skoky 192.168.1.2, 192.168.1.1, 192.168.0.1. Posledným skokom je monitorací

server.

Identifikovať sa na ceste podarí dvoch Agentov, a to 192.168.1.2 ako Agent1 a 192.168.0.1 ako Monitoring Server. Monitoring server sa podarí identifikovať na základe informácie, že je vždy posledný.

Obsah súborov na serveri môžeme vidiet na diagrame 4.3.



Obr. 4.3: Diagram stavu mapy siete po odoslaní informácií prvého agenta

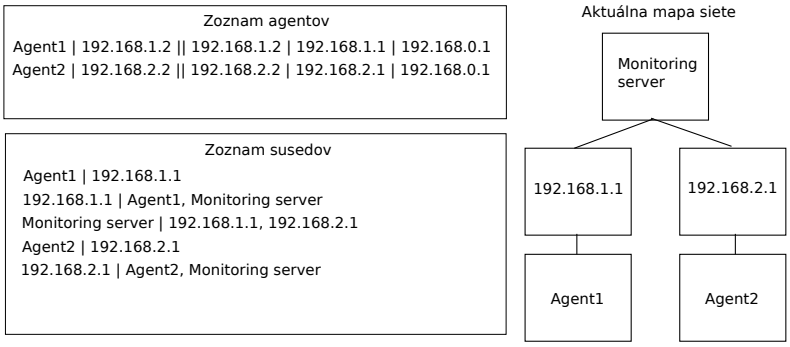
V druhom kroku odošle informáciu o stave siete Agent2. Agent2 má jedno aktívne sieťové rozhranie eth0, na ktorom má nastavenú IP adresu 192.168.2.2 a cesta predstavuje zoznam IP adries 192.168.2.2, 192.168.2.1, 192.168.0.1.

Ako vidíme, tieto informácie nemali žiaden vplyv na identifikáciu predchádzajúcich záznamov. Zoznam susedov sa teda začne tvoriť najprv od cesty Agent1 v nezmenenej podobe. Následne sa pridajú dvojce Agent2, 192.168.2.1 a 192.168.2.1, Agent2. Jediným spoločným identifikátorom je Monitoring Server. Monitoring Server obsahuje dvoch susedov 192.168.2.1 a 192.168.1.1

Na tomto príklade si môžme všimnúť anomáliu algoritmu pre vytváranie siete. Algoritmus zdanlivo nesprávne spočítal, že agent Hop sú v skutočnosti dva počítače. Situácia sa však zmení, ak odošle informáciu aj inkriminovaný agent Hop. V prípade, že by nebol agentom, ale iba sieťovým prvkom, ostala by mapa siete zobrazená tak, ako to môžme vidiet na diagrame 4.4.

Predpokladajme teda, že program agent je nainštalovaný aj na skoku jedna. V tomto prípade agent odošle všetky potrebné informácie o stave siete na monitorovací server.

Pri identifikácii sieťových rozhraní už nevystupujú IP adresy, ale iba mená v sieti. Začneme teda podľa algoritmu vytvárať jednotlivé dvojce. Vytvorenie

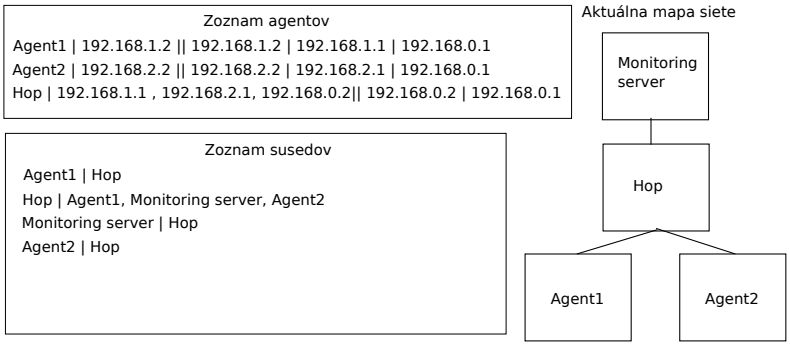


Obr. 4.4: Diagram v druhom kroku

dvojíc z informácií od prvého agenta sa líši iba tým, že sa identifikuje adresa 192.168.1.1.

Vytváranie dvojíc z informácií od druhého agenta pridá riadok Agent2 a Hop. Následne sa IP adresa 192.168.2.2 identifikuje ako Hop so susedmi Agent2 a Monitoring server. Nakoľko Hop, už je v zozname susedov uvedený, nevytvára sa nová tabulka, ale monitorovací server sa pokúsi informácie pridať do zoznamu susedov agenta Hop. Pridanie Agent2 prebehne podľa očakávaní. Monitoring server sa však nepridá, nakoľko jeho uvedenie v zozname by vytvorilo duplicitu. Podobne sa už nič nezmení na susedoch agenta Monitoring server. Pridanie by spôsobilo duplicitu.

Ako vidíme v tomto prípade sa na diagrame 4.5 zobrazí celá mapa siete.



Obr. 4.5: Diagram v treťom kroku

4.5 Ukladanie dát na serveri

Okrem vytvárania dát má server za úlohu aj ukladať dáta od jednotlivých agentov. Agenti posielajú dáta v pravideľných intervaloch a zväčša ide o rôznorodé dáta v ktorých sa vyhľadáva podľa jediného kľúča.

Spôsob uloženia dát by mal ukladať dáta úsporne, bez zbytočných redundancií. Dáta z jednotlivých agentov majú pevnú veĺkosť, preto by sa dali uložiť do databázy. Ako príklad môže slúžiť databáza MYSQL, Oracle či PostgreSQL. Inštalácia takejto databázy zaberá pomerne veľa miesta a vyžadovala by si externú závislosť na knižnici pre pripojenie k databáze.

Vyhľadávanie v databázi prebieha sekvečne v prípade, že nad dátami nie je vytvorený index. Index tvorí zvyčajne druh stromu v závislosti od použitého engine. Takýto index teda tvorí redundanciu uložených dát.

Nakoľko môžme predpokladať, že vyhľadávanie v dátach bude prebiehať iba podľa času, rozhodol som sa pre uloženie dát do utriedeného súboru. Vyhľadávanie dát v tomto súbore prebieha v logaritmickom čase, teda rovnako, ako pri intervalovom vyhľadávaní v databáze.

Uloženie do súborov je rozdelené podľa jednotlivých agentov v adresári špecifikovanom v konfiguračnom súbore doplnené o meno modulu. Cesta k súboru teda začína /storage/directory/agent_name/module_name. Súbory v adresári si ukladá každý modul samostane.

Ukladanie do súboru je realizované pomocou funkcie *store_data* a *select_data*.

Funkcia *store_data* dostáva ako parameter meno súboru a odkaz na dáta. Dáta sú uložené v súbore s hlavičkou na začiatku. Hlavička reprezentuje štruktúru a veľkosť vyhľadávaných dát. Funkcia skontroluje formát zadaných dát. V prípade, že časová známka je vyššia ako časová známka posledného záznamu, funkcia dáta odmietne uložiť. Návratovou hodnotou je počet uložených dát.

Vyhľadávanie nad dátami je dané parametrom *time_from*, *time_to* a *desired_count*. Funkcia si najprv vyhľadá časové známky *time_from* a *time_to* v súboru binárnym pôlením. Binárne pôlenie vráti offset[1] v súbore na začiatok dát v logaritmickom čase vzhľadom na počet záznamov.

[1]Počet bytov v súbore od jeho začiatku

So znalosťou informácii offset-ov hraničných časov, server vyhľadá horný celý násobok žiadaných dát a tie namapuje do pamäti.

4.6 Utilita viewgenerator, vytváranie redundancií

K programu je prípojená taktiež utilita viewgenerator. Utilita slúži na vytváranie náhľadov s menším rozlíšením, ako je rozlíšenie dané agentom. Utilita je kompilovaná spoločne so serverom.

Vytvorenie dát v novom rozlíšenie sa zaistí spustením utility s parametrami -i, označujúci vstupný súbor -o označujúci súbor výstupný, -r nastaví nové časové rozlíšenie. Príklad volania uvádzam vo vloženom kóde 4.3.

Listing 4.3: Príklad volania utility viewgenerator

```
./viewgenerator -i input_file -o output_file \
   -r new_resolutiom -t time_to -r
```

Utilita pri vytváraní náhľadu uzamkne pôvodný náhľad pomocou funkcie *flock*, teda počas vytvárania náhľadu sú nedostupné dáta pre zápis.

Algoritmus na vytvorenie nového náhľadu je pomerne jednoduchý. Pomocou funkcie *file_range_data*, agent vyberie dáta a žiadané offset-y v logaritmickom čase. Následne agent prejde všetky dáta a priebežne si ukladá rozdiel v uložených dátach. V prípade, že rozdiel prekročí zadané rozlíšenie dáta zapíše.

Okrem funkcie vytvárania náhľadov funkcia taktiež poskytuje výpis dát pomocou parametru -p.

4.7 Diagnostikovanie problémov v sieti

Server okrem funckie ukladania dát ponúka možnosť informovať administrátora siete na problémy. Nastavenie informovanie spočíva v uvedení parametru time_alert v konfiguračnom súbore.

Server si po pripojení agenta do spojového zoznamu uloží aktuálny čas. Súčasne pri spustení zavolá samostatné vlákno, ktoré pristupuje k spojovému

zoznamu. Nakoľko je prístup zdieľaný, použil som spojový zoznam chránený mutex-om.

Informovanie priebeha pomocou odosielania e-mailu. Pre odoslanie e-mailu musí mať server nainštalovaný program mail, ktorý server spustí volaním *popen*. Ako text emailu sa uvedie čas, ktorý sa agent neozval.

V práci je implentované aj informovanie užívateľa prostredníctvom sms. K odosielaniu sms je použitá platená sms-brána sms-brana.sk. Užívateľ môže špecifikovať prihlasovacie meno prostredníctvom parametrov sms_username a sms_password v konfiguračnom súbore.

Odosielanie sms je implementované pomocou utility wget a sms-api na sms-brana.sk. Program spustí program wget, ktorý zostaví spojenie HTTP na server.

Kapitola 5

Klient

Klient je jednoduchý zobrazovací program napisaný v programovacom jazyku C++. Slúži na zobrazanie dát, ktoré mu poskytuje server. Program implementuje prácu so sieťou, zobrazuje mapy siete a na zobrazovanie veličín používa obdobu modulov podobne ako agent a server, avšak na strane klienta sú napevno prikompilované k spustiteľnému súboru.

5.1 Qt Framework

Tvorba GUI rozhrania prebiehala v prostredí Qt. Medzi najväčie výhody prostriedia Qt patrí jednoduchá prenositeľnosť medzi operačnými systémami, podmienená iba pritomnosťou Qt Framework a preklompilovaním aplikácie pod zdrojovou platformou. Bezospornou výhodou je taktiež široká podpora dátových štruktúr (QVector, QList, QMap ..), knižnica neblokujúca prácu so sieťou (QTcpSocket) a unifikovaný prístup k SQL databázam.

Interktívne prepájanie jednotlivých prvkov zabezpečuje Qt Signal and Slot System [14], čím sa Qt Framework líši od väčšiny iných Framework-ov. Každá trieda, ktorá obsahuje kľúčové slovo *Q_OBJECT*, obsahuje okrem statí *private*, *public* a *protected* state *public slots*, *private slots* a *signals* . Takýto objekt je pred kompiláciou predspracovaný do tzv. moc súboru, na ktorý je následne volaný kompilátor.

Účelom signálov a slotov je poskytnúť systém prepojenia objektov, ktorý sa

bude aktivovať na udalosti v systéme, akými sú stlačenie tlačidla, nepravidelné príchody dát zo siete a pod.

5.2 Konfigurácia

Klient na svoju konfiguráciou používa knižnicu libsqlite3, pomocou triedy implementovanej v *Database*. Trieda *Database* vytvára sqlite3 databázu v aktuálnom adresári.

Trieda *Database* je singleton, čo znamená, že môže byť instanciovaná práve raz. Prístup k instancii poskytuje funkcia *SQLITE_Database::instance*. V databázi sa ukladá zoznam monitorovacích serverov a ich autentifikačné údaje.

5.3 Zobrazenie mapy siete

O zobrazenie mapy siete sa stará trieda *Networkmap*. Konštruktor triedy vyšle signál na triede Network, ktorá odošle serveru informáciu o poskytnutí dát pre zobrazenie mapy siete.

Jednotlivé skoky sa v sieti ukladajú vo widgete *Hopwidget*, ktorý obsahuje ukazatele na ostatných susedov reprezentovaných v triede *Hopwidget*. V zozname susedov nie je odkaz na otca, teda prvý skok na ceste od agenta k serveru.

Tak ako bolo popísané v časti o serveri, dáta zo serveru reprezentuje zoznam susedov, ktorý trieda *Networkmap* prevedie na stromovú štruktúru podľa nasledujúceho algoritmu.

V priebehu algoritmu si program pamätá dve množiny. Množinu ešte nezobrazených skokov v sieti a množinu skokov sieti, ktoré už boli zobrazené. Program postupuje v cykle pokiaľ nie je množina nezobrazených skokov prázdna. Z množiny nezobrazených prvkov sa vyberie prvý nepridaný prvok a v ďalšom kroku sa pozrie na jeho susedov. Môžu nastať tri prípady.

- V prípade, že sa nevyskytujú v množine spracovaných prvkov, instanciuje sa trieda *Hopwidget* a pridajú sa do zoznamu, ktorý čaká

na spracovanie. Súčasne si ukazatele pridáme do zoznamu svojich synov.

- V prípade, že sa vyskytujú v množine už spracovaných prvkov a nie sú našim otcom, pridá sa ukazateľ, ktorý bol získaný z množiny do zoznamu susedov aktuálneho prvku.
- V prípade, že je skok naším otcom program pokračuje v ďalšom kroku.

Algoritmus zostaví štruktúru siete od monitorovacieho serveru k agentom. Pokiaľ v zozname spracovaných agentov nie sú všetci agenti, znamená to, že na ceste od agenta k serveru existoval aspoň jeden router, ktorý neodpovedal ICMP packetom na požiadavok utility traceroute.

Mapa siete sa vykreslí algoritmom na prechod stromu s použitím fronty, teda BFS [2]. Miernou odlišnosťou od pôvodného algoritmu je vytvorenie zoznamu už vykreslených skokov s ktorými sa spája aktuálne vykreslovaný skok v prípade ak v sieti existoval cyklus.

Na začiatku sa do fronty pridá monitorovací server. V každom kroku algorimtu sa pridá do fronty susedov (v ktorých sa z prechádzajúceho algoritmu nevyskytoval otec). Každý prvok si pamätá ľavý a pravý okraj okna, kam môže kresliť, pričom sám sa vykreslí do stredu tohto rozsahu. Svojím synom, ktorí ešte nie sú v zozname vykreslených skokov, rozdelí priestor na rovnomerné časti.

Vykreslovanie prebieha nastavením vlastnosti *Geometry*. Ide teda o manuálne nastavený *QLayout*. Alternatívou by bolo implementovanie vlastného správcu derivovaním triedy *QLayout*.

5.4 Práca so sieťou a prepojenie s modulmi

O pripojenie k sieti sa stará trieda *Network*. Trieda *Network* je súčasťou Widget-u ServerWidget, ktorý po reakcii na užívateľov vstup inicializuje pripojenie na sieť. Na prepojenie používa TCP sockety implementované v triede *QTcpSocket*.

Pripojenie v klientskej aplikácii je odlišné od pripojenia v časti serverovej. Pre beh klienta je potrebné iba jedno vlákno, ktoré však musí interaktívne

reagovať na vstupy od užívateľa, presúvanie a zmenu okien. Už z prvotného pohľadu je jasné, že pripojenie užívateľa v pomalej sieti by spôsobovalo zbytočne dlhé reakcie čakania na užívateľov vstup.

Riešenie poskytuje priamo Qt Framework využitím signálov a slotov pre interakciu so sieťou. Riešenie je zdanlivo podobné riešeniu pomocou systémového volania *poll*, popísaného v predchádzajúcich kapitolách. Klient si pri konštrukcii objektu správy siete zaregistruje do systému slotov ukazateľ na metódu triedy. Táto metóda je volaná pokiaľ sa na serveri objavia dáta.

Nakoľko k sieti pristupuje viacero komponent súčasne, je potrebné vyriešiť identifikovanie prijatých dát. Takéto dáta môžu buď patriť modulom, alebo sú súčasťou nadväzovania spojenia a patria teda sieti.

Sieť funguje ako stavový automat so 5 stavmi. Pokiaľ sieť nebola aktivovaná nachádza sa v stave 0. Po úspešnom pripojení a prijatí dát zo servera prejde sieť do stavu 1. Následne odošle požiadavku na autorizáciu, čím prejde do stavu 2. Pokiaľ bola autorizovaná úspešne, žiadne ďalšie dáta, ktoré odošle server klientovi, jej už nepatria.

Posledné dva stavy sú vyhradené pre odosielanie požiadavkov modulmi. Jednotlivé moduly odosielajú svoje požiadavky sieti zavolaním metódy, ktorá pridá požiadavku do zoznamu evidovaných požiadaviek. Táto metóda vráti volajúcemu unikátne číslo požiadavky.

Stav 3 je vyhradený na čas, kedy sieť nečaká na prijatie žiadnych dát v serveri. Pokiaľ klient prejde do stavu 3 a vo fronte požiadaviek sa nachádza aspon jeden prvok, dáta z fronty požiadaviek odošle a prejde do stavu 4. Druhou možnosťou je, že dáta vo fronte neboli. V tom prípade sa obsluhujúca funkcia skončí.

Modul teda po pridaní dát nezmení stav sieťového manažéra, iba zavolá obslužnú rutinu, podobne ako by ju spustil signál. Sieť následne odošle požiadavku z fronty, pokiaľ nebola prázdna.

Pokiaľ je zavolaný handler v stave 4, sieť očakáva prijatie dát. Sieť dáta prijme a notifikuje o tom všetkých účasníkov. Dáta patria tomu, koho identifikátor dostali. Po úspešnom prijatí dát prejde späť do stavu 3.

5.5 Zobrazenie grafov

O zobrazovanie grafov sa stará knižnica Qwt. Zobrazenie grafu prebieha v triede *GraphWidget*, ktorá obsahuje komponenty na nastavovanie času a triedu *Plot*. Trieda vznikla deriváciou z triedy *QwtPlot*. Trieda implementuje obsahuje jednu vykreslovaciu krivku *Curve*, ktorá čerpá dáta z triedy *TimedData*, ktorá vznikla deriváciou triedy *QwtData*.

V prípade zmeny rozlíšenia trieda emituje signál o zmene dát. Signál prijme nadradená trieda (modul). Modul o nové dáta požiada triedu *TimedData* zostavením požiadavku struct *client_module_request*. Pokiaľ prijme dáta, spracuje ich a odošle pre zobrazenie, zostaví vektor štruktúr *timed_data* a skonštruje graf.

Modul v ponímaní klienta teda slúži ako prostredník medzi sieťou a grafom. Z modulov, ktoré zobrazujú dáta som implementoval *LibDiskusage*, *LibTraffic* a *LibCpuusage*.

Kapitola 6

Porovnanie monitorovacích systémov

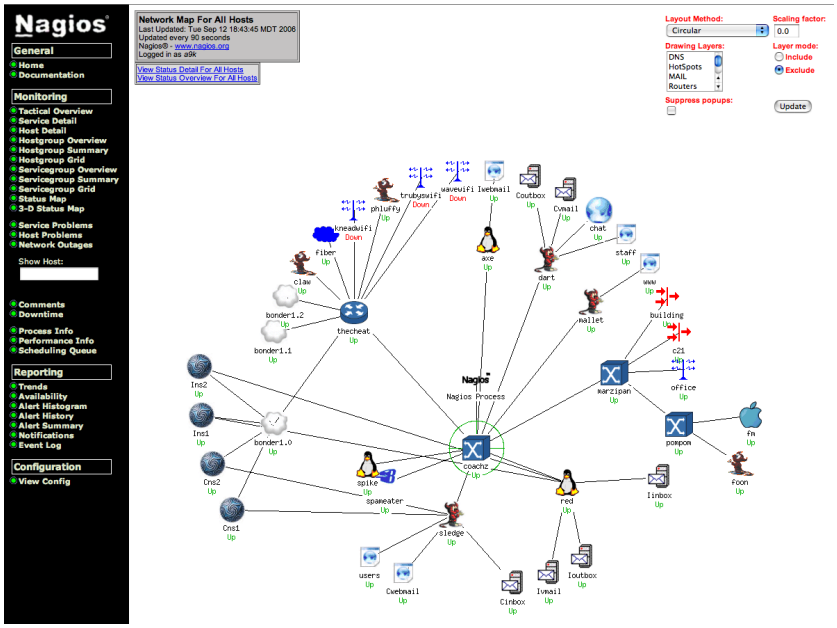
6.1 Nagios

Program Nagios je najrozšírenejším programom na správu siete. V roku 2010 mal 250000 aktívnych používateľov[6]. Je veľmi dobre škálovateľný, no pomerne ťažko konfigurovateľný. Nagios sa integruje do sieti podobne ako program popísaný v tejto práci a to inštaláciou na centrálny server.

Typická konfigurácia systému Nagios spočíva väčšinou v definovaní jednotlivých bodov, ktoré má systém monitorovať. Monitorovanie prebieha pomocou utilít ping, traceroute a pomocou protokolu SNMP[10] a utility telnet. Pomocou utility telnet dokáže monitorovať funkčnosť serverov ako FTP, HTTP, SQL či SMTP. Podobne ako môj program používa systém zásuvných modulov.

Pre svoj beh vyžaduje väčšinu z programov, ktoré sám monitoruje a to HTTP server Apache, SMTP server Postfix a SQL Server.

Architektúra programu je podobná ako architektúra programu implementovanom v tejto práci. Implementuje agentov, ktorí zbierajú dáta o jednotlivých prvkoch v sieti. Narozdiel od programu centrálna jednotka Nagios-u sa zväčsa sama aktívne dotazuje na jednotlivé prvky v sieti a pokiaľ nie je nejaký dotaz v súlade s užívateľsky definovanými funkciami, informuje



Obr. 6.1: Záber na program Nagios.

užívateľa najčastejšie mailom.

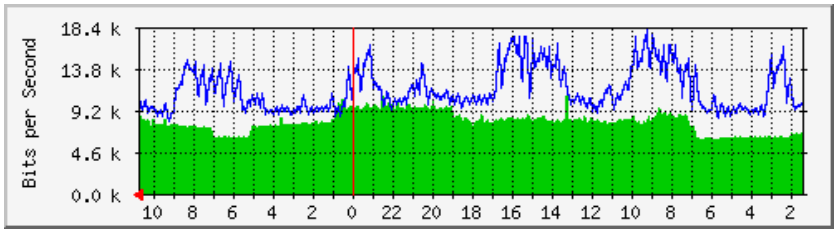
Nagios nepodporuje SMS API, na druhej strane však dovoľuje užívateľovi nakonfigurovať si vlastné funkcie na informovanie.

Program Nagios neimplementuje grafického klienta, ovláda sa pomocou webovej aplikácie.

6.2 MRTG

Program MRTG [7] monitoruje sieťovú prevádzku na jednotlivých sieťových spojeniach. Neposkytuje žiadne interaktívne mapy, avšak generuje HTML

stránky, ktoré obsahujú prehľady o zaťažení na jednotlivých častiach sieťe. Sieťové zariadenia monitoruje pomocou protokolu SNMP.



Obr. 6.2: Graf z MRTG na sieti kolej.mff.cuni.cz.

Program je napísaný v jazyku Perl a podporuje operačné systémy Unix/Linux, Windows a existuje aj port na operačný systém MacOS.

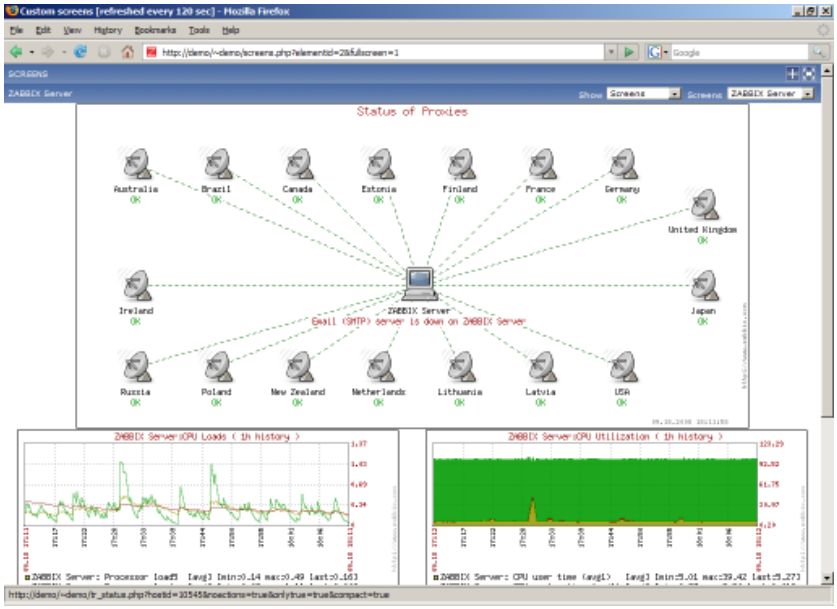
MRTG je iba informačným systémom, ktorý zbiera dáta a používa sa často v spojení so systémom Nagios. Pokiaľ sa spolu s programom použije HTTP server, sú jeho výstupy dostupné aj cez sieť internet.

6.3 Zabbix

Zabbix [18] podobne ako Nagios ponúka komplexný systém na monitorovanie sietí. Architektúrou je podobný systému implementovanom v bakalárskej práci, implementuje agent-server model. Zabbix sa skonfiguruje nainštalovaním zabbix-agentov na jednotlivé spojenia. Inštalácia je pomerne kompikovaná, no existuje množstvo tutoriálov a online dokumentácie.

K dispozícii sú balíčky pre operačné systémy Windows, Linux/GNU a FreeBSD. Zabbix ukladá dáta na serveri do relačnej databázy MySQL, PostgreSQL alebo Oracle. Zobrazovanie prebieha pomocou programu zabbix-frontend a dynamicky generuje HTTP stránky na internetovom serveri. Preto je pre zobrazovanie a ukladanie dát zrejmá závislosť SQL servera a servera HTTP.

Administrácia Zabbix prebieha cez webové rozhranie, podporuje užívateľské listy práv ACL. Okrem získavania dát od agentov dokáže monitorovať aj sieťové rozhrania pomocou protokolu SNMP. Vykreslovanie grafov pre-



Obr. 6.3: Zobrazenie agentov v programe Zabbix.

bieha pomocou pridania nových komponent, ktoré toto zobrazenie podporujú. Dokáže taktiež nakresliť jednoduchú mapu siete.

Zabbix ponúka množstvo nastavení pre upozornenie užívateľa. Dokáže informovať pomocou e-mailov, SMS, či správ na Jabberi. Jedná sa v súčastnosti o najkomplexnejší systém. K jeho prednostiam patrí aj malá záťaž na procesor a používanie RAM. Oproti programu prezentovanom v práci je však patrná závislosť na veľkom množstve knižníc.

Kapitola 7

Záver

Zámerom práce nebolo vyvinúť monitorovací systém podobných kvalít ako sú dlho vyvíjané projekty Nagios či Zabbix, ale ponúknuť prehľad, ako je možné monitorovať metropolitné bezdrôtové siete v súčastnosti a navrhnúť dizajn distribuovaného programu, ktorý je schopný získať veličiny relevatné pre chod metropolitnej siete.

V predloženej práci som implementoval radu algoritmov, ktoré ponúkajú úspornejšie využitie systémových prostriedkov ako programy podobného zamerania. Implementovaný systém dokáže s malými nárokmi na systémove prostriedky monitorovať stredne veĺkú počítačovú sieť a informácie o jednotlivých zariadeniach ukladať na pevný disk počítača. Program je ľahko konfigurovateľný a odolný proti výpadkom spojenia. Zobrazovací klient zároveň ponúka základný prehľad o jednotlivých parametroch siete s pohľadom do minulosti.

Kvôli rozsahu projektu sa mi z časových dôvodov nepodarilo implementovať mnohé zaujímavé vlastnosti, ako napríklad podporu šifrovania pri odosielaní dát, štatistiku pri zbieraní dát, či podporu pre viaceré operačné systémy najmä na strane agenta. O tieto vlastnosti by som chcel program v neskoršej dobe rozšíriť.

Literatúra

[1] Brian Hall: *Beej's Guide to Network Programming* , Jorgensen Publishing, 2009.

[2] *Breadth First Search*
http://en.wikipedia.org/wiki/Breadth-first_search

[3] *Embedded System*
http://en.wikipedia.org/wiki/Embedded_system

[4] *File Descriptor*
http://dictionary.die.net/file%20descriptor

[5] *Liblcfg*
http://liblcfg.carnivore.it/

[6] *Nagios*
http://www.nagios.com/products/nagiosxi

[7] *MRTG*
Tobi Oetiker's MRTG - The Multi Router Traffic Grapher

[8] *Optické pripojenie na kolene*
http://ronja.twibright.com/

[9] *Pthread mutex*
http://opengroup.org/onlinepubs/007908775/xsh/pthread_mutex_lock.html

[10] *RFC1157 - Simple Network Management Protocol (SNMP)*
http://www.faqs.org/rfcs/rfc1157.html

[11] *Sqlite3 - lightweighted database*
http://www.sqlite.org/

[12] *Qt Reference Documentation*
http://doc.trolltech.com/4.6/index.html

[13] *Qt Supported platforms*
http://doc.qt.nokia.com/4.6/supported-platforms.html

[14] *Qt Signal and slots*
http://doc.trolltech.com/4.6/signalsandslots.htmll

[15] *Qwt - Qt Widgets for Technical Applications*
http://qwt.sourceforge.net/

[16] *C++ Singleton design pattern*
http://www.yolinux.com/TUTORIALS/C++Singleton.html

[17] *The Single UNIX Specification, Version 2 - regex.h*
http://opengroup.org/onlinepubs/007908799/xsh/regex.h.html

[18] *Zabbix*
http://www.zabix.com

Dodatok A

Obsah priloženého CD

Cd priložené k bakalárskej práci obsahuje zdorojové kódy k programu. V jednotlivých priečinok sú obsiahnuté príkladové konfiguračné súbory a konfiguračné súbory vývojového prostredia KDevelop. Nakoľko program môže obsahovať chyby, aktuálnu verziu je možné stiahnuť pomocou príkazu.

```
svn co http://tdo.sk/svn/asc
```

Dodatok B

Krátka užívateľská príručka

B.1 Kompilácia

Agent, server a klient sa skompilujú pomocou príkazu cmake v adresári so zdrojovými kódmi. Ako parameter programu cmake zadávame meno konfiguračného súboru CMakeLists.txt. Utilita cmake vygeneruje súbor Makefile. Následná kompilácia prebehne zadaním príkazu make.

Po skompilovaní nájdeme spustiteľné súbory v adresári build. Pre skomplilovanie klienta je nutné nainštalovanie framework-u Qt a pre podporu grafov taktiež framework Qwt a nastavenie premennej LD_LIBRARY_PATH na užívateľskú knižnicu Qwt. Knižnica Qwt sa po rozbalení skompiluje príkazmi qmake a make.

Zdrojový kód B.1 zozbrazuje stiahnutie aktuálnych zdrojových kódov.

Listing B.1: Stiahnutie a skompilovanie agenta a serveru

```
svn co http://tdo.sk/svn/asc

cd asc/agent/
cmake CMakeLists.txt
make

cd ../server
cmake CMakeLists.txt
make
```

B.2 Spustenie a konfigurácia - agent

Agent sa spúšta príkazom ./agent uvedeným v zdrojovom kóde B.2.

Listing B.2: spustenie agenta

```
./agent -c meno_konfiguracneho_suboru
```

Konfiguračný súbor mení jednotlivé nastavenia agenta. Základnými parametrami sú

- server_name - meno servera,
- server_port - port, kam sa prihlasuje server,
- cache_file - meno súboru pre dočasné ukladanie dát,
- username - prihlasovacie meno,
- password - prihlasovacie heslo,
- modules_directory - meno priečinku, kde bude agent vyhladávať moduly,
- send_time - ako často má agent odosielať dáta.

Ďalšie nastavenia sa týkajú jednotlivých modulov. Konfigurácia modulu prebieha tak, že sa z jeho mena odstráni prípona .so a pridá sa konfiguračný parameter. Pre libtraffic.so by konfiguácia času znamenala pridanie konfiguračného parametru libtraffic_time.

Pre úspešné načítanie modulu do pamäti je nevyhnutné uviesť aspoň jeden konfiguračný parameter a to jeho interval _time, ktorý nastavuje, ako často sa má spúštať meranie veličiny modulom.

Nepovinným parametrom je parameter _thread, ktorý určí, či sa má modul bežať v samostatnom vlákne. Druhým parametrom je _arg, ktorý je predávaný modulu pri štarte.

Príklad konfiguračného súboru uvádzam v kóde B.3 pre zavedenie merania voľného miesta na disku a monitorovania mapy siete s koncovým bodom definovaným IP adresou 92.240.244.54.

Listing B.3: Konfiguračný súbor agenta

```
server_name = ”92.240.244.55”
server_port = ”4401”
modules_directory = ”./modules_enabled/”
cache_file = ”/tmp/monitor-agent/cache”
username = ”agent1”
password = ”google”
send_time = ”10”
libnetworkmap_time = ”2”
libnetworkmap_arg = ”92.240.244.54”
```

B.3 Spustenie a konfigurácia - server

Server sa spúšta príkazom ./server ako je uvedené v príkaze B.4.

Listing B.4: spustenie serveru

```
./server -c meno_konfiguracneho_suboru
```

Konfiguračný súbor je podobný ako v prípade agenta

- server_port - port, kde počúva server,
- agent_auth_list - zoznam s autentifikačnými údajmi agentov,
- client_auth_list - zoznam s autentifikačnými údajmi klientov,
- modules_directory - cesta k modulom,
- storage_directory - priečinok na ukladanie dát,
- server_greeting - správa, ktorú odosiela server po prihlásení agentovi.

Pre pripojenie jednotlivých modulov ich stačí vložiť do priečinku špecifikovanom v konfiguračnom súbore zadanom možnosťou modules_directory. V konfiguračnom súbore B.5 vidíme monitorovací server, ktorý počúva na porte 4401.

Listing B.5: Konfiguračný súbor serveru

```
server_port = ”4401”
```

```
modules_directory = "./modules_enabled/"
storage_directory = "/tmp/server-asa/"
agent_auth_list = "agent_auth"
client_auth_list = "client_auth"
server_greeting = "Monitoring server v0.1b"
```

B.4 Spustenie klienta

Nakoľko klient je interaktívnou aplikáciou, jeho spustenie je pomerne priamočiare. Klient sa spúšta príkazom ./client, pričom treba mať správne skonfigurovanú systémovú premennú LD_LIBRARY_PATH tak, aby obsahovala dynamicky linkované knižnice Qwt. Po inštalácii knižnice qwt sa uložia v /usr/local/qwt-5.2.1-svn/lib/.

Pripojenie sa na server prebieha pridaním nového serveru do konfiguračného nastavenia. Kliknutím na connect sa zobrazí monitorovací server a jeho pripojení agenti a moduly, ako je napríklad mapa siete, ktoré nie sú závislé od agentov.

Zobrazovanie veličín prebieha rozkliknutím agenta, kde sa zobrazia moduly, ktoré boli aktivované v nejakom časovom období na serveri. Dáta k ním sa zobrazia v prípade, že je modul aktívny v dobe pripojenia klienta.