Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Teoretické informatiky

Bakalářská práce

Systém pro konfiguraci a monitorování distribuovaného systému NEMEA

Marek Švepeš

Vedoucí práce: Ing Tomáš Čejka

13. května 2014

Poděkování

Rád bych velmi poděkoval vedoucímu práce za pomoc a odborné vedení při vývoji modulu a také při psaní této práce. Zároveň chci poděkovat organizaci CESNET, z.s.p.o. za možnost být součástí tohoto projektu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či spracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2014 .....

Odkaz na tuto práci

Švepeš, Marek. *Systém pro konfiguraci a monitorování distribuovaného systému NEMEA*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstract

This thesis deals with the design and the implementation of a configuration and monitoring module, called Supervisor, for the Nemea system. Nemea is a system for analysis of network flows and anomaly detection. The system is composed of several independent interconnected software modules. Supervisor allows to manage user-created configuration of modules. The main task of Supervisor is a monitoring of Nemea modules and system load. Based on the collected information, Supervisor can decide whether it is advantageous to distribute the load among multiple modules or even multiple physical machines. The monitored features include statistics of usage of CPU and communication interfaces. Supervisor can be run in an interactive mode or as an operating system service running in background. The background process can be controlled using a thin client application created for this purpose. After series of user tests, Supervisor was successfully deployed on the server of CESNET, z.s.p.o. and in May 2014 it will be included in the new version of the Nemea package.

Keywords Configuration system, monitoring system, Nemea, system load distribution, network flows, graph data structures.

Abstrakt

Tato práce se zabývá vývojem konfiguračního a monitorovacího systému (dále jen supervizor) pro systém Nemea. Nemea je systém pro analýzu síťových toků a detekci anomálií skládající se ze softwarových modulů. Supervizor umožňuje spravovat uživatelem vytvořenou konfiguraci modulů. Hlavním úkolem supervisora je sledování stavu modulů a zatížení systému. Na základě sbíraných informací je supervisor schopen rozhodnout, zda-li je výhodné rozložit zátěž jednoho modulu mezi více modulů nebo dokonce i na více fyzických strojů. Mezi sbírané informace patří statistiky o provozu na komunikačních rozhraních modulů a využití procesoru. Program supervisor je možné spustit v interaktivním režimu nebo jako službu operačního systému na pozadí. Proces na pozadí je následně možné ovládat pomocí tenké klientské aplikace, vytvořené speciálně pro tento účel. Po sérii uživatelských testů byl supervizor úspěšně nasazen do testovacího provozu na serveru organizace CESNET, z.s.p.o. a v květnu 2014 bude začleněn do nové verze balíku systému Nemea.

Klíčová slova Konfigurační systém, monitorovací systém, Nemea, rozložení zátěže systému, toky v sítích, grafové datové struktury.

Obsah

Úvod **1**
Cíl bakalářské práce 2
Seznam požadavků na funkcionalitu 2

1 Analýza a návrh **3**
1.1 Systém Nemea 3
1.2 Orientovaný graf 4
1.3 Tok v síti 4
1.4 Stávající řešení 5
1.5 Programovací jazyk 6
1.6 Architektura modulu 6
1.7 Efektivní rozložení zátěže 12
1.8 Dostupné operace 14

2 Realizace **17**
2.1 Změny v knihovně libtrap 17
2.2 Použité struktury 18
2.3 Implementace funkcí 19
2.4 Dokumentace programu 30

3 Testování **31**
3.1 Testování programu při vývoji 31
3.2 Nasazení supervizora a jeho testování 31
3.3 Testování použitelnosti 32
3.4 Simulace zatížení modulu 33

Závěr **35**
Shrnutí průběhu práce, vlastního přínosu a výsledků práce 35
Budoucí rozšiřování 36

Literatura **37**

A Seznam použitých zkratek **39**

B Obsah přiloženého CD **41**

C Instalační manuál **43**
C.1 Instalace modulu 43
C.2 Spuštění modulu 43

D Použité datové struktury **45**

E Obrázkové přílohy **47**

Seznam obrázků

1.1 Distribuovaný systém Nemea 4
1.2 Orientovaný graf 5
1.3 Posloupnost úkonů servisního vlákna 8
1.4 Rozložení zátěže CPU 13
1.5 Rozložení zátěže při přetížení modulu 15

2.1 Vnitřní reprezentace modulů orientovaným grafem 19
2.2 Příklad výstupu funkce *Show graph* 23

3.1 Výsledek simulace zatížení modulu 34

E.1 Výsledek simulace zatížení modulu 47
E.2 Výsledek simulace zatížení modulu 48
E.3 Výsledek simulace zatížení modulu 49
E.4 Ukázka výstupu uživatelské nápovědy programu 50
E.5 Ukázka zapojení modulů před dekompozicí 51
E.6 Ukázka zapojení modulů po dekompozici 52

Úvod

Konfigurace komplexních systémů skládajících se z mnoha částí může být pro uživatele náročná a nepřehledná. Uživatel často nemá přehled o všech částech systému najednou a nemá mnoho možností, jak systém monitorovat. Jednotlivé části musí uživatel samostatně spouštět, zastavovat a kontrolovat zda běží.

Tyto problémy řeší konfigurační a monitorovací nástroje. Umožňují uživateli spravovat všechny části systému najednou a poskytují mu pokročilejší funkce jako je monitorování toku dat v systému. Vstupem bývá obvykle konfigurační soubor, ve kterém je inicializační nastavení systému spolu s nastavením prvků. Konfigurační systémy často ušetří uživateli mnoho času u terminálu, protože všechny operace jsou usnadněné a provedené v několika krocích.

Jedním z takových komplexních systémů je modulární systém Nemea pro analýzu síťových toků a detekci anomálií vyvíjený v rámci organizace CESNET z.s.p.o. Systém se skládá ze softwarových modulů, z nichž každý má svou funkci. Tato práce se zabývá vývojem konfiguračního a monitorovacího systému nazývaného supervizor pro systém Nemea.

Supervizor využívá technologie XML k definování vstupní konfigurace modulů, která je pro uživatele přehledná a snadno upravitelná. Nad touto konfigurací umožňuje supervizor uživatelům provádět operace spuštění a zastavení modulu, zobrazení stavu modulů, spuštění dodatečné konfigurace a jiné.

Vedle těchto konfiguračních operací dále supervizor moduly automatizovaně monitoruje. Pravidelně si obnovuje statistiky o provozu na komunikačních rozhraních modulů a kontroluje tím tok dat v systému. Na základě těchto statistik dokáže detekovat možné přetížení modulů. Dále monitoruje využití CPU jednotlivými moduly a kontroluje, zda nedochází k jeho přetížení. Je-li procesor přetížen, dokáže supervisor moduly migrovat mezi fyzickými stroji.

Supervizor může být spuštěn v interaktivním režimu nebo v režimu procesu na pozadí, ve kterém nevyžaduje další interakci s uživatelem. V tomto režimu uživatel supervizora ovládá pomocí klienta vytvořeného pro tento účel.

Cíl bakalářské práce

Cílem této bakalářské práce je vytvořit konfigurační a monitorovací systém (dále jen supervizor), který bude spravovat a monitorovat modulární systém Nemea (více v sekci 1.1). Supervizor umožní uživatelům vytvoření centrální konfigurace modulů, se kterou bude poté pracovat a monitorovat ji.

Výsledkem práce bude funkční konfigurační a monitorovací systém běžící jako služba operačního systému, ke které se budou moci uživatelé opakovaně připojovat pomocí supervizor klienta.

Seznam požadavků na funkcionalitu

Na základě konzultací s členy organizace CESNET, z.s.p.o. (dále jen CESNET), která je zadavatelem této práce, jsou definovány tyto požadavky na funkcionalitu supervizora:

- načíst konfiguraci modulů z konfiguračního souboru,
- spouštět a ukončovat moduly,
- sledovat stav běžících modulů a v případě potřeby je znovu spustit,
- monitorovat komunikaci mezi jednotlivými moduly,
- efektivně rozložit zátěž systému mezi fyzické stroje,
- monitorovat tok zpráv v systému,
- spustit dodatečnou konfiguraci za běhu.

Kapitola 1

Analýza a návrh

1.1 Systém Nemea

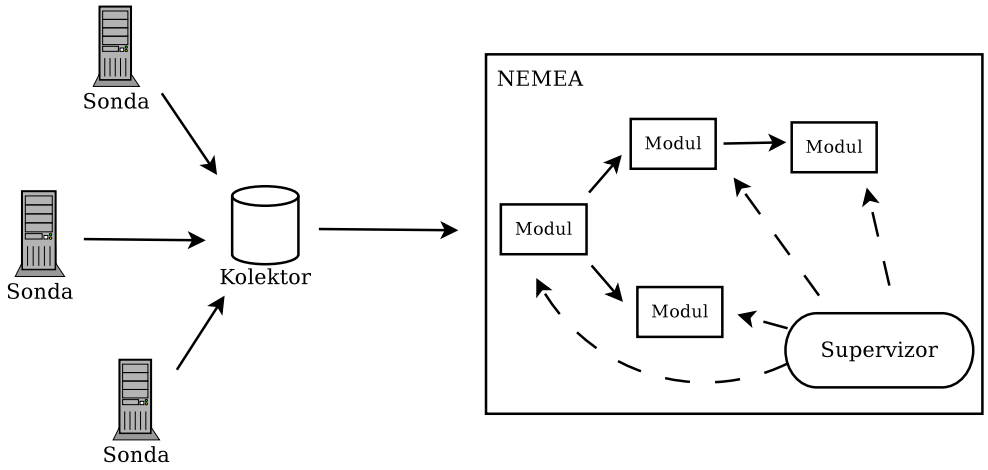
Nemea je modulární distribuovaný systém pro analýzu síťového provozu zaměřený na detekci anomálií. Je vyvíjen v rámci CESNETu a běží pod GNU/linux. Systém se skládá z paralelně běžících modulů, které si mezi sebou předávají zprávy s daty. Každý modul má nějakou specifickou funkci a dohromady vykonávají požadovanou činnost [3].

Modul je spustitelný binární soubor, který se dá spustit ručně. Ruční ovládání je ale náročné a nepraktické.

Komunikaci mezi moduly zajišťují rozhraní implementovaná v knihovně libtrap, která je součástí instalačního balíku systému Nemea. Rozhraní jsou jednosměrná, a proto jsou rozdělena na vstupní a výstupní. Výstupní rozhraní vykonává roli serveru a vstupní rozhraní roli klienta, který se k serveru připojuje. Každý modul má podle své potřeby definovaný počet vstupních a výstupních rozhraní.

Moduly podle funkce dělíme na tři hlavní typy. První typ je zdroj dat. Přes tyto moduly se do systému dostávají informace. Ty jsou převedeny do formátu, který jsou schopné zpracovávat další moduly. Druhým typem jsou moduly filtrační a agregační. Ty mají za úkol informace v systému mezi moduly rozdělovat nebo naopak od více modulů data sloučit. Mohou také podle předem daných podmínek data filtrovat a rozdělovat mezi ostatní moduly. Třetím typem jsou moduly detekční, které v přijatých datech detekují nežádoucí jev a výsledkem je kladná nebo záporná zpráva o výskytu jevu [7].

Monitorování sítě probíhá mimo systém Nemea. Monitorovací sondy získávají data o provozu v síti, která odesílají kolektorům. Tato data jsou následně proudově analyzována systémem Nemea. Monitorování počítačové sítě znázorňuje obrázek 1.1. Více o systému Nemea na [1].



Obrázek 1.1: Distribuovaný systém Nemea pro analýzu síťového provozu a detekci anomálií. Skládá se ze samostatných modulů – paralelně běžících prvků a supervizoru – ovládacího prvku. Jako zdroje dat slouží měřící sondy a kolektory. Tato práce se zabývá řídícím modulem supervizor.

1.2 Orientovaný graf

V teorii grafů je orientovaný graf G definován jako $G = \langle H, U, \rho \rangle$. H je množina hran grafu, U je množina uzlů grafu a ρ je incidence grafu definovaná $\rho : H \to U \times U$. Jedná se o množinu uspořádaných dvojic. Na obrázku 1.2a je vidět ukázka orientovaného grafu kde $H = \{h1, h2, h3, h4, h5, h6\}$, $U = \{u, v, w, x, y\}$. Příkladem incidence může být $\rho(h1) = (u, v)$. Uzel u je počáteční uzel hrany h a uzel v je koncový uzel hrany h. Zároveň je uzel u předchůdce uzlu v a uzel v je následník uzlu u [6].

Pro naše použití orientovaného grafu budeme potřebovat přiřadit jednotlivým hranám hodnotu a dostaneme tak orientovaný ohodnocený graf zobrazený na obrázku 1.2b.

1.3 Tok v síti

Pro definování toku v síti si nejdříve musíme definovat síť $S = \langle G, q, s, t \rangle$:

- graf $G = \langle H, U \rangle$ je orientovaný graf
- $q : H \to Z^+$ je kapacita hran, $q(u, v) = q_{uv}$
- $s \in U$ je zdroj sítě S, $t \in U$ je spotřebič sítě S

Poté můžeme definovat tok v síti S jako přiřazení $f : H \to Z$ splňující podmínky [5]:

1. $0 \leq f(u,v) \leq q(u,v)$ pro všechny hrany $(u,v) \in H$

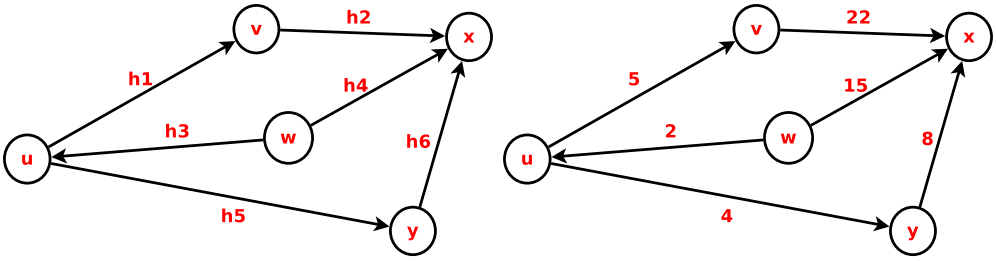
2. $\sum_{(u,v)\in H} f(u,v) - \sum_{(u,v)\in H} f(v,u) = 0$ pro všechny $u \in U$

Druhá podmínka pro náš tok v síti (tok mezi moduly v systému Nemea) nemusí platit vždy, protože rozdíl je v některých případech nulový, kladný a někdy záporný. Příčin tohoto stavu je několik:

1. V naší síti negeneruje tok pouze zdroj sítě, ale i některé vnitřní uzly. Příkladem může být modul, který přijatý tok zduplikuje.

2. V průběhu měření toku v síti může dojít ke zpoždění, které způsobí, že zprávy odeslané jedním modulem nedorazí včas ostatním modulům.

3. Poslední příčinou, která nás zajímá nejvíce je ztrátovost zpráv. Když modul X posílá zprávy modulu Y a modul Y je nestíhá všechny zpracovat, modul X přebývající zprávy zahazuje. Monitorováním ztrátovosti všech hran grafu detekujeme možné přetížení jednotlivých modulů.



(a) Orientovaný graf (b) Orientovaný ohodnocený graf

Obrázek 1.2: Příklad orientovaného grafu tvořeného množinou hran $H = \{h1, h2, h3, h4, h5, h6\}$ a množinou uzlů $U = \{u, v, w, x, y\}$. Po přiřazení hodnoty jednotlivým hranám vzniká orientovaný ohodnocený graf.

1.4 Stávající řešení

V průběhu vývoje systému Nemea byl vytvořen provizorní skript s konfiguračním souborem . Tento skript uměl jen základní operace a to spustit a vypnout modul. Nesplňoval tedy všechny požadavky, které se od tohoto nástroje pro správu a monitorování čekají. Zároveň byl napsán jako skript pro interpret příkazů bash(1) a postupným rozšiřováním se stal zdrojový kód neudržitelným. Žádné jiné poloautomatizované ovládání Nemea modulů tohoto typu zatím neexistuje.

1.5 Programovací jazyk

Požadavky při výběru programovacího jazyka byly rychlost, efektivita a kompatibilita se zbytkem systému Nemea. Na výběr proto bylo ze tří možností a to jazyk C, C++ nebo skriptovací jazyk Python. Jazyk C++ byl vyloučen hned na začátku, protože jeho hlavní výhodu objektového programování, tudíž využití tříd a dědičnosti nebylo pro tuto práci potřeba. Výběr mezi jazykem C a skriptovacím jazykem Python rozhodla hlavně úroveň znalostí jednotlivých jazyků a proto byl pro implementaci vybrán jazyk C.

1.6 Architektura modulu

Modul supervizor je vícevláknový program. Hlavní vlákno se stará o inicializaci paměťových prostředků, načtení konfigurace, načtení argumentů programu a hlavně o komunikaci s uživatelem. To znamená, že vypisuje nabídku dostupných operací, přijímá vstup v podobě výběru operace a následně ji provede. Druhé vlákno se nazývá servisní a bude popsáno v následující podkapitole Servisní vlákno. Třetí vlákno přijímá spojení od remote_supervizora, který je spuštěn na druhém fyzickém stroji a slouží k efektivnímu rozložení zátěže na více CPU (více v sekci 1.7.1).

Supervizor je konfigurační a monitorovací systém. Z tohoto hlediska ho můžeme rozdělit na dvě části – konfigurační a monitorovací. Konfigurační část se týká operací, které provádí uživatel. Jedná se například o přidání modulu, vypnutí modulu, zapnutí nebo vypnutí celé konfigurace. Monitorovací část je automatizovaná a není potřebný uživatelský zásah. Tuto část zajišťuje servisní vlákno.

1.6.1 Servisní vlákno

Toto vlákno slouží k monitorování modulů. V pravidelných intervalech si obnovuje stavové informace modulů, které jsou evidované v supervizoru pro každý modul zvlášť a na jejich základě provádí další úkony. Řídí se tedy zcela automatizovaně.

Mezi stavové informace modulu patří:

- stav modulu – modul běží/neběží,
- servisní rozhraní – modul byl/nebyl spuštěn se servisním rozhraním,
- připojen – supervizor je/není k servisnímu rozhraní modulu připojen,
- počet znovu spuštění – čítač počtu znovu spuštění modulu,
- obsloužen – modul byl/nebyl obsloužen servisním vláknem,
- vzdálený modul – modul byl/nebyl spuštěn na záložním fyzickém stroji,

- naklonovaný modul – modul byl/nebyl kvůli přetížení naklonován.

Stavové informace může přepisovat jen supervizor. Z pohledu uživatele jsou pouze ke čtení. Jinak je to u konfiguračních informací, které jsou z pohledu uživatele ke čtení i zápisu. Hlavní konfigurační informací je *přepínač stavu*, který určuje, zda má/nemá modul běžet.

Servisní vlákno pravidelně kontroluje stav běžících modulů, vytížení procesoru každým z nich, přetížení samotných modulů a zda-li je supervizor k modulům připojen. Posloupnost úkonů, které servisní vlákno provádí, ukazuje obrázek 1.3.

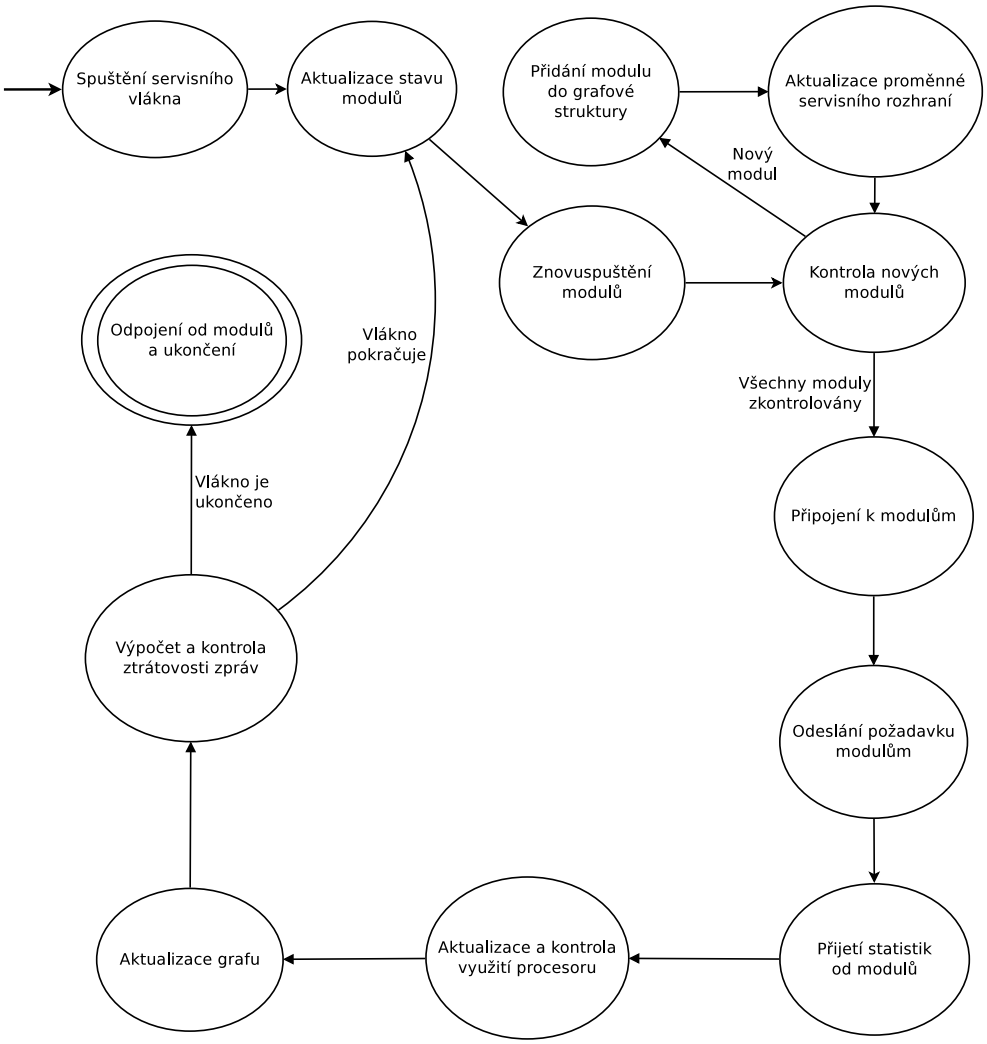
Ke komunikaci supervizora s moduly bylo vytvořeno samostatné rozhraní nazvané servisní rozhraní, aby nebyl provoz na ostatních rozhraních nijak narušen ani zpomalen. Toto servisní rozhraní slouží k získávání statistik o komunikaci mezi moduly, což je další věc, kterou servisní vlákno kontroluje. Servisní rozhraní je volitelné a moduly mohou být spuštěny i bez něj. V tom případě ale supervizor nezíská potřebné statistiky. Mezi sledované statistiky patří: počet přijatých zpráv na každém vstupním rozhraní, počet odeslaných zpráv z každého výstupního rozhraní a počet odeslání obsahu vyrovnávací paměti z každého výstupního rozhraní. Informace o provozu na komunikačních rozhraních jsou využívány pro sledování zátěže jednotlivých spuštěných modulů a k detekci možného přetížení systému. Servisní rozhraní včetně potřebných čítačů je implementováno uvnitř knihovny libtrap, která je moduly linkována při spuštění.

Servisní vlákno se po spuštění chová jako konečný automat. Všechny operace se periodicky opakují, dokud není supervizor ukončen. Automat obsahuje tyto stavy:

Aktualizace stavu modulů
: Supervizor si obnoví u všech spuštěných modulů stavovou informaci *stav modulu* podle toho, jestli stále běží proces potomka s identifikátorem běžícího procesu modulu.

Znovu spuštění modulů
: Po předchozí aktualizaci *stavu modulů* supervizor znovu spustí neběžící moduly. Spuštění se řídí podle *stavu modulu* a podle *přepínače stavu*. Je-li *přepínač stavu* povolen a modul neběží, je ihned znovu spuštěn.

Kontrola nových modulů
: Servisní vlákno kontroluje, zda-li obsloužilo všechny moduly. Neobsloužené moduly nejdříve přidá do datových struktur programu, konkrétně grafových struktur a propojí se zbylými moduly. Poté zkontroluje rozhraní modulu a zjistí, jestli bude modul spuštěn se servisním rozhraním. Tuto informaci aktualizuje v příslušné stavové proměnné *servisní rozhraní*. Jsou-li obslouženy touto operací všechny moduly, pokračuje vlákno na další operaci. Při dalším průchodu tímto stavem vlákno kontroluje pouze neobsloužené moduly.



Obrázek 1.3: Konečný automat uvnitř servisního vlákna zobrazující posloupnost úkonů automatizovaného monitorování modulů.

Připojení k modulům

Servisní vlákno projde všechny moduly a u modulů se servisním rozhraním kontroluje stavovou informaci *připojen*. Jestliže supervizor k modulu se servisním rozhraním není připojen, servisní vlákno se pokusí o připojení a výsledek operace aktualizuje ve stavové proměnné *připojen*.

Odeslání požadavku modulům

Servisní vlákno odesílá požadavek na vrácení statistik od modulů. Aby byl požadavek modulu odeslán, musí být splněny následující podmínky: modul musí běžet, musí být spuštěný se servisním rozhraním a supervizor k němu musí být připojen. Takovému modulu je odeslán požadavek

a očekává se od něj okamžitá odpověď.

Přijetí statistik od modulů
Supervizor přijme od běžících modulů se servisním rozhraním statistiky s hodnotami čítačů zpráv a odeslání vyrovnávací paměti a aktualizuje je v datových strukturách reprezentujících modul.

Aktualizace a kontrola využití procesoru
Supervizor aktualizuje vytížení CPU jednotlivými moduly za uplynulou periodu. Aktualizované informace o vytížení CPU jsou zkontrolovány a když některý modul splňuje podmínky přetěžování CPU, je zastaven a spuštěn na záložním fyzickém stroji, na kterém běží *remote_supervizor*. Moduly s ním spojené jsou přenastaveny a restartovány, aby byl systém stále propojen. Více o vytížení CPU moduly v sekci 1.7.1.

Aktualizace grafu
Supervizor všem modulům se servisním rozhraním aktualizuje podle přijatých statistik s čítači hodnoty hran, vedoucích k modulům, ke kterým je připojen v grafové struktuře programu.

Výpočet a kontrola ztrátovosti zpráv
Na základě obnovených hodnot hran grafové struktury je spočtena ztrátovost zpráv rozdílem odeslaných a přijatých zpráv každé dvojice propojených modulů. Spolu s dalšími uloženými hodnotami je poté spočten odhad střední hodnoty pro každou propojenou dvojici modulů [4]. Výsledky jsou porovnány se standardní odchylkou s nastavitelnou prahovou hodnotou. Je-li detekováno přetížení modulu, přijímající modul je naklonován a data jsou rozdělena mezi původní modul a klon pro rozložení zátěže modulu. Více o přetížení modulů v sekci 1.7.2.

Odpojení od modulů a ukončení
Když je supervizor ukončen, servisní vlákno se v tomto stavu odpojí od modulů se servisním rozhraním a ukončí se.

1.6.2 Možnosti spuštění

Supervizora je možné spustit ve dvou režimech:

Interaktivní režim
První režim byl užitečný pro vývoj a testování supervizora a je vhodný například na testování nových modulů. Jedná se o klasické spuštění programu z terminálu, kam se opakovaně vypisuje nabídka dostupných operací a uživatel zadá číslo operace, která se má provést.

Daemon-client
Druhý režim je důležitý pro reálné nasazení modulu supervizor do pro-

vozu. Jedná se o architekturu *supervizor-daemon* a s ním komunikujícího *supervizor-klienta*. Supervizor se spustí s přepínačem --daemon jako proces na pozadí a nevyžaduje další interakci od uživatele. Uživateli je umožněno se systémovým procesem *supervizor-daemon* dále komunikovat pomocí *supervisor-klienta*. Tím se může kdykoliv připojit k běžícímu procesu daemona, provést požadované akce a zase se odpojit. Toto je možné opakovat do chvíle ukončení daemona uživatelem.

Komunikace procesu *supervizor-daemon* a *supervizor-klient* probíhá přes UNIX soket. Je možné ovládat supervizora ze vzdáleného počítače a to například pomocí SSH protokolu. Uživatel se pomocí SSH připojí k počítači, na kterém běží supervizor, pomocí klienta se připojí k procesu a provede požadované operace. Princip ovládání je stejný jako u předchozího způsobu ovládání. Standardní vstup klienta je přeposlán soketem na standardní vstup procesu daeomon a naopak standardní výstup procesu daemon je přeposlán soketem na standardní výstup klienta.

1.6.3 Vstup programu

Program přijímá předdefinovanou sadu přepínačů, z nichž jsou některé povinné a některé nepovinné. Povinný přepínač je -f následovaný názvem konfiguračního souboru s počáteční konfigurací modulů. Nepovinné přepínače jsou -h, -v, --daemon a --show-cpuusage. -h přepínač zobrazí podrobnou nápovědu s funkcemi a možnostmi spuštění programu. -v je verbose přepínač, který určuje, jak moc podrobný má být výpis supervizora. Přepínač --daemon spustí program jako systémový proces na pozadí. --show-cpuusage přepínač zobrazuje průběžně vytěžování CPU jednotlivými moduly.

Konfigurační soubor je ve formátu XML, protože je strojově čitelný, přehledný a snadno upravitelný. Výpis 1.1 ukazuje konfiguraci jednoho modulu. Struktura konfiguračního souboru byla navržena po konzultaci s budoucími uživateli supervizora. Zahrnuje název modulu, cestu ke spustitelnému souboru modulu, parametry se kterými se spouští a poté seznam všech komunikačních rozhraní a jejich parametrů, se kterými bude modul spuštěn. U rozhraní je definován typ spojení TCP nebo UNIX soket, poté směr komunikace, protože rozhraní jsou jednosměrná a od směru odvíjející se parametry rozhraní. Poslední položkou konfigurace rozhraní je poznámka, která je nepovinná a slouží pro potřebu uživatelů, kteří spravují konfiguraci.

Celá konfigurace ze souboru je po spuštění supervizora parsována a načtena do vnitřních struktur programu pomocí knihovny libxml2 [2]. Tato knihovna byla použita hlavně kvůli budoucímu rozšiřování supervizora. Používají ji i aplikační vrstvy se kterými bude v budoucnu supervizor propojen.

```
<module>
  <params>-n 2</params>
  <name>traffic_merger</name>
```

```
  <path>../modules/traffic_merger/traffic_merger</path>
  <trapinterfaces>
    <interface>
      <note></note>
      <type>TCP</type>
      <direction>IN</direction>
      <params>localhost,8000</params>
    </interface>
    <interface>
      <note></note>
      <type>TCP</type>
      <direction>IN</direction>
      <params>localhost,8001</params>
    </interface>
    <interface>
      <note></note>
      <type>TCP</type>
      <direction>OUT</direction>
      <params>8002,3</params>
    </interface>
    <interface>
      <note></note>
      <type>SERVICE</type>
      <direction></direction>
      <params>service1,1</params>
    </interface>
  </trapinterfaces>
</module>
```

Výpis 1.1: Příklad obsahuje konfiguraci ve formátu XML pro spuštění modulu pro slučování toků zpráv. Modul má jedno výstupní rozhraní a dvě vstupní. Navíc se modul spustí i se servisním rozhraním.

1.6.4 Výstup programu

Výstup supervizora a spuštěných modulů je pro přehlednost a možnost zpětného dohledání statistik nebo příčin ukončení modulů rozdělen a přesměrován do různých souborů.

Standardní výstup a standardní chybový výstup jednotlivých spuštěných modulů je přesměrován do samostatných souborů se jménem modulu a označením výstupu. Tyto logy obsahují časové značky každého spuštění modulu a jsou uloženy ve složce *./modules_logs/*. Díky chybovým hlášením modulů je možné zpětně zjistit příčinu ukončení modulu.

Výstupy supervizora jsou rozděleny do tří kategorií. První kategorie je standardní výstup, který je zobrazován v terminálu. V případě spuštění v režimu procesu na pozadí je přesměrován do souboru *./supervizor_log*. Když se k supervizoru připojí klient, standardní výstup je přeposlán soketem na standardní výstup klientovi. Po odpojení se výstup opět přesměruje do stejného souboru.

Druhá kategorie jsou události modulů, mezi které patří:

- spuštění, zastavení a znovu spuštění modulu,
- povolení či zakázání modulu,
- připojení či odpojení k servisnímu rozhraní modulu,
- spuštění a zastavení celé konfigurace.

Každá z těchto událostí je opatřena časovou značkou a je zaznamenána do souboru*./supervisor_log_module_events*.

Poslední třetí kategorií jsou statistiky od modulů. Mezi ně patří čítače, které se posílají přes servisní rozhraní a také záznamy o vytížení CPU. Tyto statistiky jsou ukládány do souboru *./supervisor_log_statistics* a také obsahují časové značky.

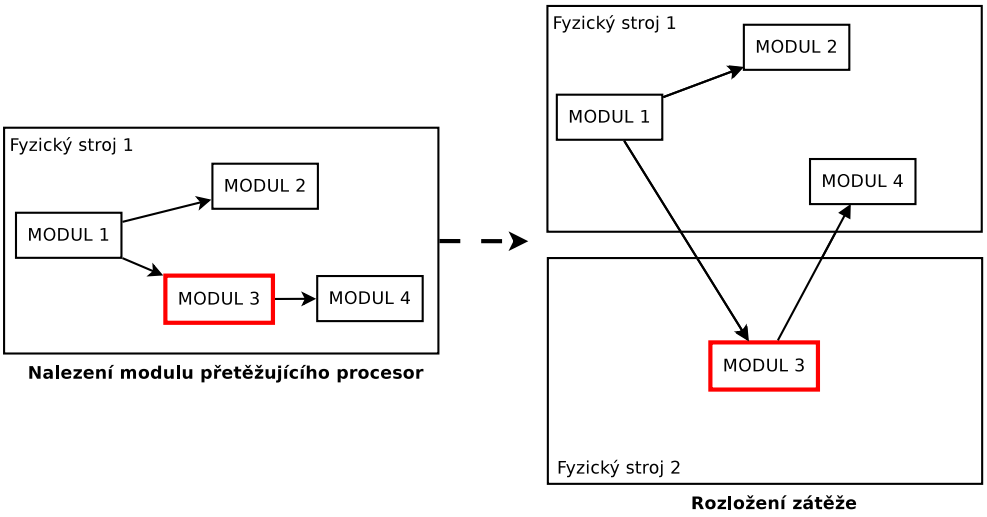
1.7 Efektivní rozložení zátěže

Supervizor monitoruje zatížení jednotlivých modulů i zatížení CPU samotnými moduly. Zatížení CPU supervizor kontroluje pomocí míry využití CPU daným procesem. Zatížení jednotlivých modulů kontroluje výpočtem ztrátovosti zpráv a následnou detekcí přetížení modulu. Předpokládáme, že pokud výstupní rozhraní zahazuje zprávy, je to proto, že vstupní rozhraní nestíhá přijímat nebo komunikační kanál není dostatečně rychlý pro přenos všech zpráv. Řešení stavu, kdy je systém přetížen, provádí supervizor několika způsoby. Jedním z nich je spuštění daného modulu na jiném fyzickém stroji a propojení zbytku systému s daným modulem. Jiným řešením je dekompozice modulu.

1.7.1 Zatížení CPU jednotlivými moduly

Supervizor monitoruje vytížení procesoru u každého modulu zvlášť a také všemi moduly dohromady. Tuto kontrolu provádí servisní vlákno jak je naznačeno na obrázku 1.3. Zatížení jedním modulem je vypočítáno poměrem využití procesoru modulem ku celkovému využití procesoru. Informace o využití procesoru modulem získává supervizor ze souboru */proc/PID/stat*, kde PID je identifikátor běžícího procesu daného modulu. Je-li dlouhodobě procesor zatěžován jedním modulem nadměrně a ostatní běžící moduly se kvůli tomu nedostanou k výpočtu, dojde k jeho spuštění na jiném fyzickém stroji jak

je naznačeno na obrázku 1.4. Tato situace nastane i tehdy, když je procesor nadměrně zatížen všemi moduly dohromady. V tom případě je vybrán modul, který ho vytěžuje nejvíce a ten je přesunut na jiný fyzický stroj.



Obrázek 1.4: Ukázka rozložení zátěže systému při nadměrném přetěžování CPU fyzického stroje jedním modulem.

Na záložním počítači je pro případ přesouvání modulů spuštěn *supervisor_remote*. Jedná se o zjednodušenou verzi supervizora, který umí modul spustit, restartovat, zastavit a monitorovat stav. Rozšířený je o připojení k supervizoru a přijímání příkazů s daty modulů. Jedná se tedy o klienta, který se připojuje k supervizoru. K přijetí spojení od *supervisor_remote* slouží dedikované vlákno supervizora.

Při situaci z obrázku 1.4 je detekován modul 3 jako modul, který nadměrně zatěžuje procesor, a proto je výhodné ho přesunout. Supervizor tedy modul zastaví, přenastaví mu rozhraní a pošle příkaz *supervisor_remotu* ke spuštění spolu s daty tohoto modulu. Poté přenastaví rozhraní modulů, které se připojují k přemístěnému modulu a restartuje je. Tím se znovu automaticky propojí systém.

1.7.2 Přetížení modulu

Běžící moduly jsou reprezentovány orientovaným, ohodnoceným grafem. Základním předpokladem je jejich spuštění se servisním rozhraním. Každý vrchol obsahuje rozhraní, se kterými byl spuštěn. Výstupní a vstupní rozhraní dvou různých modulů se stejným portem jsou spojena orientovanou hranou. Například jeden modul obsahuje výstupní rozhraní s portem 8000 a druhý modul vstupní rozhraní s portem 8000. Orientovaná hrana tedy povede z výstupního rozhraní do vstupního. Propojená rozhraní umožňují rychlé efektivní procházení grafu a kontrolování toku zpráv v systému.

Detekce přetížení modulu
Systém Nemea data zpracovává proudově. Proto, když jeden modul nestíhá zpracovat data včas, nesmí docházet ke zpoždění celého systému. Moduly tedy odesílají zprávy i v případě, že k nim připojené moduly tyto zprávy nepřijímají, a proto jsou zahazovány. Důvodem je neblokující odesílání a časové limity. Tento stav nastane v případě, že modul nestíhá zpracovávat množství zpráv, které mu přicházejí. Detekci přetížení modulu provádí servisní vlákno v pravidelných intervalech, jak je naznačeno na obrázku 1.3. U každého uzlu grafu představujícího modul se kontroluje výstupní rozhraní a kam z něj vede orientovaná hrana. Poté spočte rozdíl odeslaných a přijatých zpráv rozhraní, která jsou danou hranou spojena a aktualizuje odhad střední hodnoty a standardní odchylku. Tyto výpočty jsou provedeny pro každou hranu grafu zvlášť. Překročí-li aktuální rozdíl zpráv odchylku o nastavený povolený limit, je detekováno výrazné přetížení modulu.

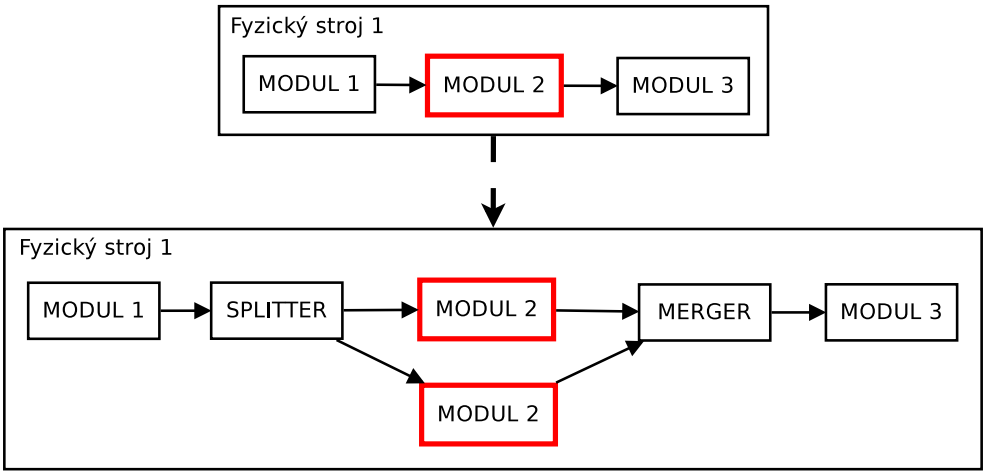
Řešení přetížení modulu
Supervizor řeší přetížení modulu dekompozicí modulu. Přetížený modul je naklonován a pomocí speciálních modulů *Splitter* a *Merger* spojen se zbytkem systému. *Splitter* je modul, který rozděluje přijaté zprávy na vstupním rozhraní rovnoměrně mezi klienty připojené na výstupní rozhraní. *Merger* naopak všechny zprávy přijaté na vstupním rozhraní od více modulů slučuje dohromady a posílá je na jedno výstupní rozhraní. Přetíženému modulu, klonu a modulům, které se na přetížený modul připojují jsou přenastavena rozhraní a moduly jsou restartovány. Toto řešení znázorňuje obrázek 1.5.

1.8 Dostupné operace

Na základě požadavků na funkcionalitu supervizora byl vytvořen seznam veřejných funkcí, které se volají v hlavní funkci a vykonávají požadované operace. Tyto veřejné funkce obsahují volání ostatních interních funkcí.

Inicializace *api_initialization()*
Inicializace je volána ihned po spuštění programu. Zajišťuje parsování parametrů programu, inicializaci a alokaci paměťových prostředků. Dále také zajišťuje načtení počáteční konfigurace ze souboru. Podle načtených parametrů programu rozhoduje v jakém ze dvou zmiňovaných režimů program poběží.

Spuštění konkrétního modulu *api_start_module()*
Na základě dalšího vstupu od uživatele je spuštěn konkrétní modul z konfigurace. Před provedením této funkce je zkontrolováno, zda již daný modul neběží. V průběhu této funkce se vytváří soubory pro standardní



Obrázek 1.5: Ukázka rozložení zátěže při přetížení jednoho modulu. Modul je spuštěn dvakrát a data jsou pomocí modulu *Spliter* rozdělena mezi přetížený modul a klon. Po zpracování v oddělených výpočetních větvích jsou data sloučena dohromady pomocí modulu *Merger*.

výstup a standardní chybový výstup a jsou do těchto souborů přesměrovány. Pro potřeby monitorování modulu jsou alokovány zbylé paměťové prostředky.

Vypnutí vybraného modulu *api_stop_module()*
Tato funkce zobrazí běžící moduly a po dalším uživatelském vstupu jeden konkrétní zastaví. Tomuto modulu je aktualizována konfigurační informace *přepínač stavu* na 0. Modul tudíž nebude servisním vláknem znovu spuštěn. Informace o modulu zůstávají z důvodu možné pozdější potřeby uloženy ve strukturách programu.

Spuštění celé konfigurace *api_start_configuration()*
Tato funkce je nadstavbou funkce *Spuštění konkrétního modulu*. Důvodem je ušetření operací při spouštění všech načtených modulů. Operace projde všechny moduly a použije u nich funkci *Spuštění konkrétního modulu*.

Vypnutí celé konfigurace *api_stop_configuration()*
Tato funkce je nadstavbou funkce *Vypnutí vybraného modulu*. Důvodem je ušetření operací při vypínání všech běžících modulů. Operace projde všechny běžící moduly a použije u nich funkci *Vypnutí vybraného modulu*.

Nastavení enabled příznaku modulu *api_set_enabled()*
Tato funkce umožňuje nastavit konkrétnímu modulu konfigurační informaci *přepínač stavu*, která rozhoduje o tom, jestli je modul v případě

potřeby znovu spouštěn, čili jestli má běžet. Funkce je nutná po vypnutí některého modulu uživatelem.

Zobrazení stavu spuštěných modulů *api_show_module_status()*
Tato funkce zobrazí všechny dostupné moduly a jejich stav. Moduly jsou ve stavu běžící nebo zastaven.

Zobrazení načtené konfigurace *api_show_configuration()*
Tato funkce přehledně zobrazí načtenou konfiguraci z XML souboru. Zobrazí jméno modulu, cestu k binárnímu souboru, parametry modulu a potom všechny jeho rozhraní. U každého rozhraní opět podrobně vypíše zmiňované parametry směr, typ, parametry a nepovinnou poznámku.

Zobrazení grafu běžících modulů *api_show_graph()*
Po zavolání této funkce dojde k zobrazení grafu běžících modulů se servisním rozhraním. Tato operace se skládá z několika podoperací. Nejprve je vygenerován kód grafu, ze kterého je následně vytvořen programem dot(1) obrázek ve formátu PNG a ten je zobrazen programem display(1) za běhu.

Spuštění dodatečné konfigurace *api_run_temp_conf()*
Tato funkce umožňuje za běhu supervizora přidat modul do konfigurace. Vložený XML kód s konfigurací modulu je parsován a přidán k ostatním načteným modulům do struktur programu. Po načtení je přidaný modul ihned spuštěn a monitorován servisním vláknem.

Ukončení supervizora *api_quit()*
Tato funkce ukončí celý program včetně spuštěných modulů. Nejprve je zastaveno servisní vlákno, které se odpojí od běžících modulů se servisním rozhraním. Poté jsou zastaveny moduly a na závěr jsou uvolněny veškeré paměťové prostředky supervizora.

O tyto funkce se stará hlavní vlákno na základě zadaných příkazů uživatelem.

Kapitola 2

Realizace

2.1 Změny v knihovně libtrap

Aby bylo možné monitorovat moduly systému Nemea, bylo potřeba rozšířit funkcionalitu knihovny libtrap. Díky tomuto řešení automaticky přibyla podpora monitorování všech modulů bez nutnosti moduly jakkoliv upravovat.

Implementace statistik spočívala v přidání čítačů rozhraním modulu. Každé rozhraní modulu má své vlastní čítače. Vstupní rozhraní má čítač přijatých zpráv. U výstupních rozhraní jsou to pro každé rozhraní tři samostatné čítače a to čítač odeslaných zpráv, čítač odeslání vyrovnávací paměti a čítač mechanismu, který zajišťuje automatické odeslání obsahu vyrovnávací paměti. Každý typ čítačů je tvořen polem čísel, přičemž počet prvků v polích odpovídá počtu vstupních rozhraní u čítačů přijatých zpráv a počtu výstupních rozhraní u ostatních čítačů. Přidány tedy byly alokace a uvolnění příslušných paměťových prostředků. Při provedení příslušné operace (odeslání zprávy, přijetí zprávy nebo odeslání vyrovnávací paměti) se inkrementuje prvek pole daného typu čítače s indexem pořadí rozhraní.

Na straně modulu bylo ke komunikaci se supervizorem využito v knihovně implementované výstupní rozhraní. Výstupní rozhraní představuje server a supervizor klienta, který se k němu připojuje. Komunikace probíhá na lokálním stroji, zvolil jsem proto UNIX soket. Na straně supervizora jsem tedy u servisního vlákna, které komunikuje se servisním rozhraním implementoval klientskou část, která se připojuje na servisní rozhraní modulu.

Dále bylo přidáno do libtrap k servisnímu rozhraní zvláštní vlákno zajišťující přijetí požadavku od supervizora a následné odeslání všech čítačů. Vzhledem k tomu, že obě strany dopředu znají velikost bloku paměti, který se posílá, mohou být poslány všechny čítače poskládané za sebou v souvislém bloku. Vlákno používá pro přijetí požadavku neblokující funkci *recv()*, aby se v případě ukončování modulu stihlo samo ukončit.

Proces získávání statistik tedy funguje následovně: supervizor spustí modul se servisním rozhraním a ihned poté se k němu připojí. V tu chvíli se u vlákna

servisního rozhraní spustí cyklus na odesílání statistik. Servisní vlákno supervizora (více v podsekci 1.6.1) v pravidelných intervalech odešle požadavek modulu. Ten ho přijme a reakcí je odeslání čítačů statistik.

2.2 Použité struktury

Všechny supervizorem používané struktury jsou vypsány v příloze D.

2.2.1 Struktury reprezentující modul

Výpis D.1 obsahuje strukturu *running_module_s*, která v programu reprezentuje modul. Kvůli rychlému přístupu k jednotlivým modulům jsem zvolil uložení těchto struktur do pole. Ve struktuře jsou uloženy všechny stavové proměnné a také v poli uložené struktury s načtenou konfigurací rozhraní modulu. Tato struktura *interface_s* je zobrazena na výpisu D.3. Položky struktury odpovídají definici rozhraní v konfiguraci.

Kvůli neznámé velikosti všech položek konfiguračního souboru je paměť alokována dynamicky při parsování. To se týká zmiňovaných struktur *interface_s*, řetězců v těchto strukturách a dále *module_name*, *module_path* a *module_params* řetězců v *running_module_s* struktuře.

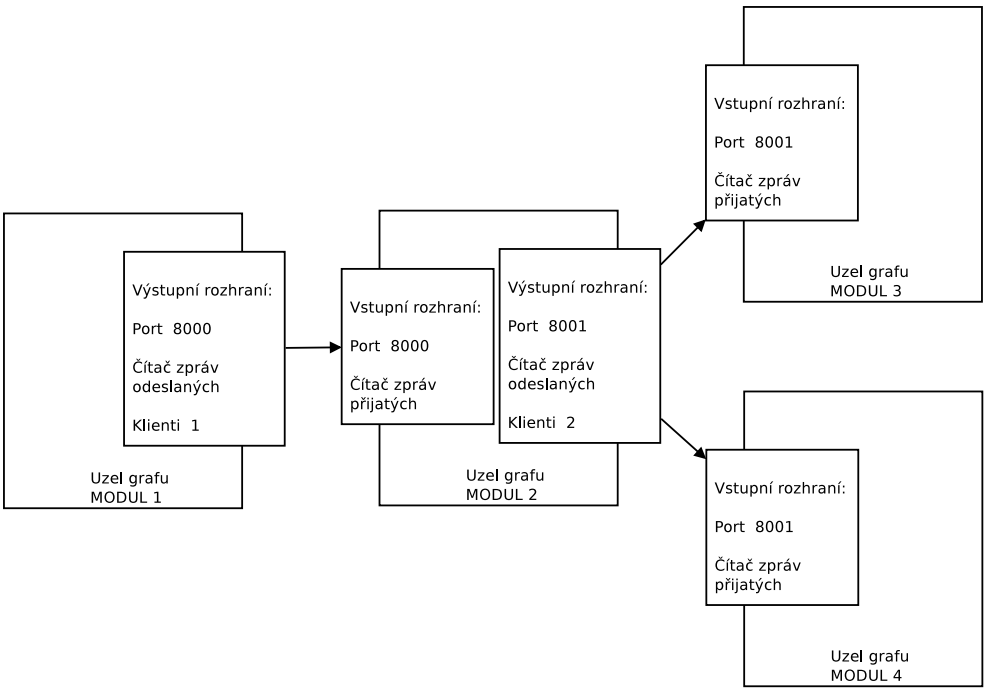
2.2.2 Struktury reprezentující uzel grafu

Po načtení je každý modul přidán do grafové struktury programu. Jedná se o orientovaný ohodnocený graf, který reprezentují struktury na výpisech D.4, D.5 a D.6. Uzel grafu je reprezentován strukturou *graph_node_s*. Většina operací s grafem je prováděna nad všemi uzly, proto byly uzly uloženy do jednosměrného spojového seznamu. Struktura *graph_node_s* proto obsahuje ukazatel na další uzel *next_node*.

Každý uzel dále obsahuje ukazatel na vstupní rozhraní a výstupní rozhraní, která jsou reprezentována každé svou strukturou. Důvodem je rozdílnost potřebných proměnných obou struktur. Orientace grafu je z výstupního rozhraní do vstupního, tudíž struktura reprezentující výstupní rozhraní obsahuje ukazatele na vstupní rozhraní. Těch je naalokováno takové množství, které je schopná daná serverová část přijmout. Struktura vstupního rozhraní obsahuje ukazatel na rodiče, v tomto případě na celý uzel. Tyto ukazatele umožňují bez problému procházet graf. Z výstupního rozhraní se lze ukazatelem dostat na vstupní rozhraní připojeného modulu, to obsahuje ukazatel na celý uzel (modul) a z něho se zjistí ukazatele na další připojené moduly. Struktura grafu je zobrazena pro představu na obrázku 2.1.

2.2.3 Ostatní struktury

Na výpisu D.7 je zobrazena struktura *edge_statistics_s*, která je použita při výpočtech ztrátovosti zpráv (více v sekci 2.3.2.6). Poslední je struktura *remote_info_s* zobrazena na výpisu D.2, která je použita k posílání příkazů *remote_supervizoru* (více v sekci 2.3.2.5).



Obrázek 2.1: Ukázka vnitřní reprezentace modulů orientovaným grafem.

2.3 Implementace funkcí

Supervizor je rozdělen na část konfigurační (uživatelskou) a část monitorovací. Každou část provádí samostatné vlákno. Konfigurační část je závislá na uživatelském vstupu, zatímco monitorovací část je automatizovaná. Obě části ale mezi sebou sdílí globální proměnné a také stavové proměnné samotných modulů. Proto bylo nutné využít synchronizačních prostředků, konkrétně mutexů, abych předešel přepisování proměnných. O konfigurační část se v obou režimech spuštění stará hlavní vlákno. Před provedením operací *Spuštění a vypnutí konkrétního modulu, Spuštění a Vypnutí celé konfigurace, Nastavení enabled příznaku modulu a Spuštění dodatečné konfigurace* si hlavní vlákno zamkne mutex a odemkne ho až po provedení celé operace. Servisní vlákno, které se stará o monitorovací část je v tu chvíli zablokované. Když je mutex volný, servisní vlákno daný mutex zamkne, provede celý monitorovací cyklus,

který je zobrazen na obrázku 1.3 a mutex opět odemkne. Poté se na několik sekund uspí a tento proces opakuje do ukončení supervizora.

2.3.1 Implementace uživatelských funkcí

Jedná se o funkce zmiňované v sekci 1.8, které obsahují interní funkce. Vykonávají uživatelem vyžadované operace s moduly.

2.3.1.1 Funkce "Spuštění konkrétního modulu"

Od uživatele je očekáván vstup s vybraným modulem, který se má z načtené konfigurace spustit. Zadané číslo je index do pole struktur *running_module_s*.

Nejprve jsou inicializovány stavové informace modulu ve struktuře *running_module_s* včetně alokace pole statistik *module_counters_array*. Poté je zavolána funkce *fork()* pro vytvoření nového procesu. Rodičovský proces v případě úspěchu nastaví stavovou proměnnou *stav modulu* na 1. V procesu potomka se nejdříve vytvoří ve složce *./modules_logs/* soubory pro standardní výstup (stdout) a standardní chybový výstup (stderr) a pomocí funkce *dup2()* se přesměruje stdout a stderr procesu do těchto souborů. Soubory mají jména ve formátu *číslo modulu_jméno modulu_výstup*.

Poté už pro spuštění binárního souboru stačí zavolat funkci *execvp()*. Moduly systému Nemea ale přijímají různé parametry z příkazové řádky, takže je nutné je připravit. Příkladem spuštění jednoho z modulů, jehož XML konfigurace je vidět v sekci 1.6.3, je tento příkaz:
./traffic_merger -i "ttts;localhost,8000;localhost,8001;8002,3;service1,1" -vvv -n 2.

Pro spuštění modulů je používána funkci *execvp()*. Prvním argumentem funkce je cesta k binárnímu souboru, která je uložena v proměnné *module_path* a druhým argumentem je pole řetězců zakončené nulovým ukazatelem. Tyto textové řetězce jsou parametry, se kterými funkce *execvp()* program spustí. Vytvoření druhého argumentu zajišťuje funkce *make_module_arguments()*, jejíž návratová hodnota je *char ***, který je poté předán funkci *execvp()*. Následující dva odstavce popisují, jak funkce *make_module_arguments()* vytvoří druhý argument při spouštění příkazu z ukázky.

První řetězec je jméno modulu, podle kterého se bude jmenovat běžící proces. Druhý řetězec je přepínač *-i* pro knihovnu libtrap. Jedná se o specifikátor komunikačního rozhraní (interface). Dalším řetězcem jsou parametry všech rozhraní modulu. Ty se skládají z části definice typů rozhraní a z druhé části s parametry jednotlivých rozhraní. Typ rozhraní je buď TCP (v řetězci tedy *t*), UNIX soket (v řetězci *u*) nebo servisní rozhraní *s*. Nejdříve jsou přidány typy všech rozhraní modulu a poté ve stejném pořadí jejich parametry, což jsou IP adresa a port v případě vstupního rozhraní nebo port a počet klientů u výstupního a servisního rozhraní. Nastavení rozhraní musí být pro knihovnu libtrap seřazeno v pořadí vstupní, výstupní a volitelné servisní rozhraní.

V případě že z konfigurace nebyl načten do proměnné *module_params* žádný řetězec, je další ukazatel nulový a funkce je u konce. V opačném případě jsou ještě po jednom přidány další parametry do výsledného pole řetězců a funkce končí.

Po přípravě parametrů je zavolána funkce *execvp()* a spustí se nový modul. V případě úspěchu funkce je modulu nastavena stavová proměnná *stav modulu* a *přepínač stavu* na 1.

2.3.1.2 Funkce "Vypnutí vybraného modulu"

Na standardní výstup jsou vypsány běžící moduly a uživatel je vyzván k zadání čísla modulu, který se má vypnout. Zadané číslo je index do pole struktur *running_module_s* běžících modulů. Funkcí *kill()* je poslán procesu s identifikátorem běžícího procesu vybraného modulu signál SIGINT. Moduly systému Nemea tento signál odchytávají, uvolní paměťové prostředky a ukončí se. Ukončenému modulu je nastavena stavová proměnná *přepínač stavu* na 0. Tím nedojde k jeho znovu spuštění servisním vláknem.

2.3.1.3 Funkce "Spuštění a Vypnutí celé konfigurace"

Díky vhodné implementaci funkcí *Spuštění konkrétního modulu* a *Vypnutí konkrétního modulu* jsou funkce *Spuštění celé konfigurace* a *Vypnutí celé konfigurace* velice jednoduché. Obě iterují přes všechny načtené moduly a použijí u nich funkci *Spuštění konkrétního modulu* při *Spuštění celé konfigurace* nebo *Vypnutí konkrétního modulu* při *Vypnutí celé konfigurace*.

2.3.1.4 Funkce "Show graph"

Funkce slouží ke snadné vizuální kontrole zapojení modulů a zobrazuje také jejich statistiky. Funkce vygeneruje ohodnocený orientovaný graf modulů.

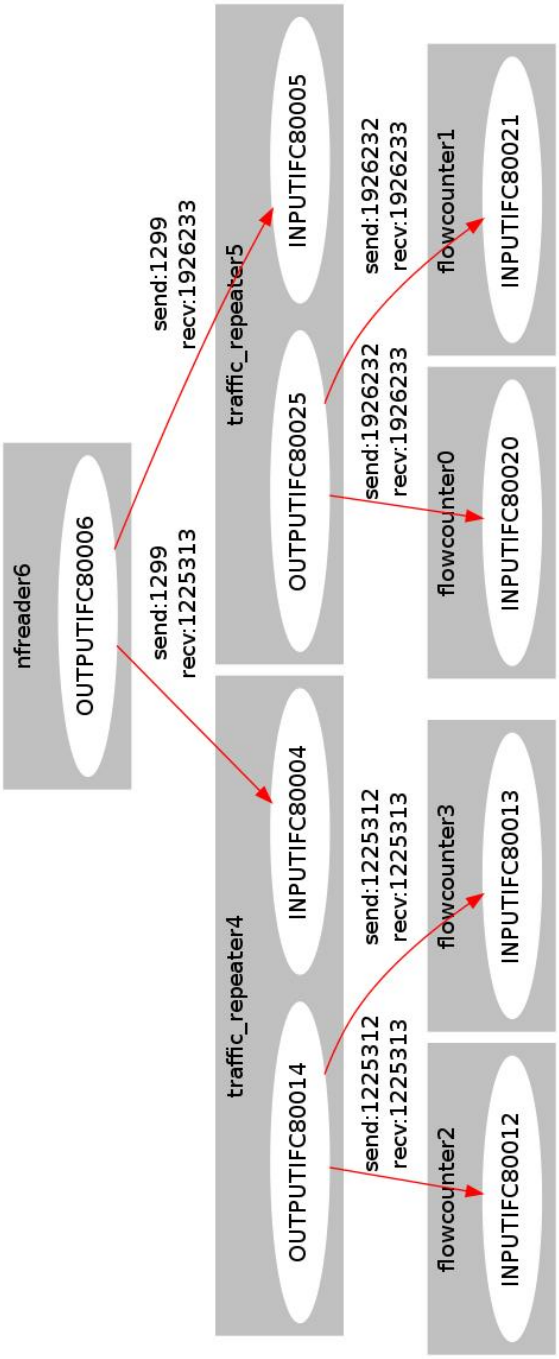
Nejdříve je vygenerován kód pro program dot(1). To se děje iterací přes všechny uzly grafu a přes jejich rozhraní. Kód je vygenerován do souboru *./graph_code*. Formát výstupu je vidět na výpisu 2.1.

Ve druhé fázi je volána interní funkce, která dvakrát po sobě volá funkci *fork()*, což vytvoří dva potomky. První potomek spustí program dot(1) a druhý program display(1). Mezi procesy spuštěných programů je vytvořena roura. Program dot(1) do ní zapisuje svůj standardní výstup a program display(1) z ní čte na svůj standardní vstup. Programu dot(1) je předán argument se souborem s vygenerovaným kódem a přepínačem pro vygenerování obrázku ve formátu PNG. Programu display(1) je zase předán argument s přepínačem, aby četl ze standardního vstupu. Výsledkem provedení celé funkce je tedy zobrazení vygenerovaného obrázku. Finální výstup je vidět na obrázku 2.2.

Pro případ, kdyby uživatel neměl nainstalován jeden ze jmenovaných programů, je do *configure* skriptu přidána podmínka, která to oznámí.

```
digraph G {
  subgraph clusterflowcounter0 {
      label="flowcounter0";
      style=filled;
      color=grey;
      node[style=filled,color=white];
      INPUTIFC80020
  }
      ...
  subgraph clustertraffic_repeater4 {
      label="traffic_repeater4";
      style=filled;
      color=grey;
      node[style=filled,color=white];
      INPUTIFC80004
      OUTPUTIFC80014
  }
      ...
  subgraph clusternfreader6 {
      label="nfreader6";
      style=filled;
      color=grey;
      node[style=filled,color=white];
      OUTPUTIFC80006
  }
  OUTPUTIFC80014->INPUTIFC80012
  [color=red, label="send:1225312\nrecv:1225313"];
  OUTPUTIFC80014->INPUTIFC80013
  [color=red, label="send:1225312\nrecv:1225313"];
  OUTPUTIFC80025->INPUTIFC80020
  [color=red, label="send:1926232\nrecv:1926233"];
      ...
}
```

Výpis 2.1: Toto je ukázka vygenerovaného kódu na základě vnitřní struktury modulů, která je reprezentována orientovaným grafem. Tento kód se poté použije pro vygenerování obrázku programem dot(1).



Obrázek 2.2: Příklad výstupu funkce, která za běhu programu vygeneruje kód pro program dot(1) na základě vnitřní struktury modulů reprezentované orientovaným grafem, vytvoří z něho obrázek a zobrazí ho programem display(1).

2.3.2 Implementace interních funkcí

Jedná se o funkce, které uživatel nepoužívá přímo. Kromě funkce *Načtení konfigurace* se jedná o funkce monitorovací části.

2.3.2.1 Funkce "Obnovení stavu a Znovu spuštění modulu"

Tyto dvě funkce následují v cyklu monitorování za sebou. Nejdříve je iterací přes všechny spuštěné moduly obnovena stavová proměnná modulů *stav modulu*. Tato informace je zjištěna pomocí funkce *waitpid()*. Argument funkce je PID procesu modulu, který je pro každý modul uložený ve struktuře *running_module_s*. Návratová hodnota funkce je 0, jestliže proces stále běží, a nebo kladná, jestliže už proces neběží.

Po obnovení stavu modulů následuje *Znovu spuštění modulů*. Když je modul povolen a neběží, dojde k jeho znovu spuštění. Tuto operaci vykonává funkce *restart_modules()*. Jedná se o stejnou funkci jako je *Spuštění konkrétního modulu* s několika změnami. Nealokují se další paměťové prostředky pro modul, protože jsou již alokované, inkrementuje se stavová proměnná modulu *Čítač znovu spuštění* a před spuštěním je provedena kontrola. U modulu může nastat problém, který znovu spuštění nevyřeší, proto je nastaven maximální počet znovu spuštění za minutu. Při jeho překročení není modul dále znovu spouštěn.

2.3.2.2 Načtení konfigurace

Tato operace je implementována ve funkci *load_configuration()*. Při načítání se parsuje XML konfigurace modulů a je uložena do struktur programu. K načtení konfigurace je použita již dříve zmiňovaná knihovna libxml2. Její funkce *xmlParseFile()* rozparsuje vstupní soubor a vrátí ukazatel na stromovou strukturu, se kterou se dále pracuje. Načtení vždy probíhá v počáteční inicializaci programu nebo za běhu v případě načítání dodatečné konfigurace. Rozdíl je v zavolání funkce knihovny libxml2 pro parsování souboru a nebo vstupního řetězce s XML konfigurací. Proto má funkce *load_configuration()* parametr, který slouží k rozlišení těchto dvou použití. Zároveň nebylo nutné zvýšit počet implementovaných funkcí veřejného programového rozhraní.

Při načítání dodatečné konfigurace je před funkcí *load_configuration()* načten vstup od uživatele. Vstup je ve formátu XML stejně jako konfigurační soubor. Ukazatel na řetězec, kam byl vstup načten, je poté předán jako argument funkci *load_configuration()*.

Jestliže je konfigurace v pořádku rozparsována knihovnou, zkontrolují se úvodní XML elementy a jestli není stromová struktura prázdná. Jestliže kontroly projdou v pořádku, jsou moduly načteny. Struktura, kterou knihovna libxml2 vytvořila, obsahuje ukazatele na potomky a sousední elementy. Soused elementu je element na stejné úrovni. Potomci elementu jsou elementy o jednu úroveň níž. Nejdříve se tedy funkce load_configuration() zanoří do

elementu *module* a načte jeho potomky *path*, *name* a *params*. Poté se zanoří o další úroveň níž do elementu *trapinterfaces* a prochází potomky s názvem *interface*. U těch načte potomky *direction*, *type*, *params* a *note*. Toto je prováděno ve smyčce dokud ukazatel na souseda není nulový, protože jsou propojeny spojovým seznamem. Pseudokód procházení struktury XML souboru zobrazuje výpis 2.2. Jedná se o čtyři vnořené cykly. První iteruje přes elementy modulů, druhý přes elementy atributů modulu, třetí přes elementy rozhraní modulu a čtvrtý přes elementy atributů rozhraní. Každý cyklus se zanořuje o úroveň níž.

Po uložení celého modulu jsou funkcí *count_module_interfaces()* zkontrolována rozhraní modulu. Jsou spočteny vstupní a výstupní rozhraní a tyto hodnoty jsou uloženy do instance struktury *running_module_s* daného modulu.

```
while (module_ptr != NULL) {
  if (module_ptr->name equals "module")
    module_atr = module_ptr->xmlChildrenNode;
    while (module_atr != NULL) {
      if (module_atr->name equals "params")
        alloc_and_save_string();
      else if (module_atr->name equals "name")
        alloc_and_save_string();
      else if (module_atr->name equals "path")
        alloc_and_save_string();
      else if (module_atr->name equals "trapinterfaces")
        ifc_ptr = module_atr->xmlChildrenNode;
        while (ifc_ptr != NULL) {
          if (ifc_ptr->name equals "interface")
            ifc_atr = ifc_ptr->xmlChildrenNode;
            while (ifc_atr != NULL) {
              if (ifc_atr->name equals "note")
                alloc_and_save_string();
              else if (ifc_atr->name equals "type")
                alloc_and_save_string();
              else if (ifc_atr->name equals "direction")
                alloc_and_save_string();
              else if (ifc_atr->name equals "params")
                alloc_and_save_string();
              ifc_atr=ifc_atr->next;
            }
          ifc_ptr = ifc_ptr->next;
        }
      module_atr = module_atr->next;
    }
    count_module_interfaces();
  module_ptr = module_ptr->next;
}
```

Výpis 2.2: Pseudokód hlavního cyklu funkce, která načítá XML konfiguraci ze souboru.

2.3.2.3 Kontrola modulů

Operace iteruje přes všechny moduly a kontroluje u nich stavovou proměnnou *obsloužen*. U nových modulů, které servisní vlákno ještě neobsloužilo, operace projde všechna rozhraní a hledá servisní rozhraní. Výsledek uloží do stavové proměnné modulu *servisní rozhraní*.

Poté je modul funkcí *add_graph_node()* přidán do grafové struktury programu vytvořením nového uzlu. Tomu jsou naalokována rozhraní reprezentovaná strukturami popsanými v sekci 2.2.2 a z konfigurace modulu jsou do těchto struktur načteny porty. Poté je prohledán zbytek grafu a jsou propojena vstupní rozhraní nového uzlu s výstupními rozhraními stávajících uzlů a naopak.

2.3.2.4 Daemon-client

Tato funkcionalita spustí supervizora jako proces na pozadí – službu operačního systému. Supervizor je v tomto režimu rozdělen na dva spustitelné soubory. První je samotná služba *supervisor* a druhý je klient *supervisor_cli*, kterým se ke službě uživatel připojí a může supervizora ovládat.

Proces na pozadí

Supervizor je spuštěn v tomto režimu po zadání parametru *--daemon* při spuštění programu. Hlavní vlákno programu v tomto případě provede inicializaci daemona místo čekání na vstup od uživatele, jak je tomu v interaktivním režimu. Pro komunikaci mezi daemonem a klientem byl použit UNIX soket, na začátku inicializace daemona je tedy vytvořen UNIX soket *supervizor_daemon.sock*. Poté je ke spuštění daemona vytvořen nový proces funkcí *fork()*. V případě úspěchu je rodičovský proces ukončen a potomek se pomocí funkce *setsid()* zařadí do nové skupiny. Rodičovský proces je poté *root*. Standardní výstup (stdout) je pomocí funkce *dup2()* přesměrován do souboru *supervizor_log* a standardní vstup (stdin) a standardní chybový výstup (stderr) jsou uzavřeny.

Po inicializaci daemona hlavní vlákno vykonává funkci serveru a čeká na připojení klienta. Je předpokládáno připojení pouze jednoho klienta, proto nebylo potřeba vytvářet samostatné vlákno na přijímání spojení. Po připojení klienta jsou v rámci klientského soketu otevřeny dva proudy. Jeden proud pro čtení a druhý pro zápis. Do těchto proudů jsou přesměrovány stdin a stdout. Místo systémových volání *send()* a *recv()* mohou být využity funkce standardní knihovny *fprintf()* pro výpis a *fscanf()* pro čtení. Obě operace probíhají v rámci soketu. Díky tomu ve zdrojovém kódu nevznikly duplicity funkcí pro načítaní nebo vypisování, které se používají v interaktivním režimu.

Poté přejde vlákno do cyklu přijímání příkazů od klienta. Použitím proudů funguje klientská část stejně jako terminál.

Když se klient odpojí, přesměruje se výstup supervizora zpět do souboru *supervizor_log*. Uzavře se klientský soket a s ním spojené proudy a hlavní vlákno čeká na opětovné připojení klienta.

Klientská část

Program se nejdříve připojí k soketu *supervizor_daemon.sock*. V případě, že nastane chyba, program se ukončí. Jinak jsou stejně jako u supervizora otevřeny v rámci soketu dva proudy. Jeden pro čtení a druhý pro zápis. Proud pro zápis slouží k odeslání uživatelského vstupu. Proud pro čtení slouží k přijetí výstupu supervizora. Čtení ze soketu i ze standardního vstupu (uživatelský vstup) je vybráno pomocí funkce *select()*. Tato funkce vrací identifikátor proudu, ze kterého je možné přečíst vstup. V tomto případě je to stdin nebo proud pro čtení ze soketu.

Po zadání vstupu uživatelem je tento vstup ihned načten do paměti programu a odeslán do soketu. Když funkce *select()* signalizuje možnost čtení ze soketu, pomocí funkce *ioctl()* je zjištěno, kolik znaků je možné přečíst a daný počet znaků je poté přečten. Návratová hodnota této funkce je počet znaků ke čtení, nebo -1 v případě chyby. Ukončení cyklu a programu nastane ve chvíli, kdy funkce *select()* signalizuje možnost čtení ze soketu a funkce *ioctl()* vrátí 0. To znamená, že supervizor ukončil spojení a uzavřel soket. Dojde k uvolnění paměťových prostředků, zavření proudů a soketu a program se ukončí.

2.3.2.5 Zatížení CPU jednotlivými moduly

Implementaci funkcionality obsahují dvě hlavní funkce *update_cpu_usage()* a *check_cpu_usage()*. První funkce aktualizuje všem běžícím modulům proměnné s vytěžováním procesoru, do zvláštní proměnné procesorový čas v uživatelském módu a do jiné proměnné v kernel módu. Tyto informace jsou získány ze souboru */proc/PID/stat*, kde PID je identifikátor běžícího procesu modulu. Soubor *stat* obsahuje 43 různých položek s informacemi o daném procesu. Pomocí funkce *scanf()* načtu potřebné dvě položky a aktualizuji je pro každý modul z tohoto souboru. Funkce *update_cpu_usage()* také aktualizuje proměnnou celkového využití procesoru všemi procesy systému, která je potřebná k výpočtu poměrného vytížení moduly. Informace o celkovém vytížení procesoru jsou v souboru */proc/stat*.

Funkce *check_cpu_usage()* nejdříve ve dvou krocích zkontroluje přetížení procesoru. První krok kontroluje dlouhodobé přetěžování jedním modulem. Vytěžuje-li modul procesor nad nastavený povolený limit, je inkrementována pomocná proměnná. Dosáhne-li tato proměnná hodnoty 10, je tento modul

vybrán pro přesun. Když při kontrole modul vytěžuje procesor pod nastavený povolený limit, je proměnná dekrementována. Minimální hodnota proměnné je 0. Jak již bylo dříve řečeno toto monitorování se periodicky opakuje a periodu kontrol řídí supervizor. Druhý krok kontroly funkce *check_cpu_usage()* sečte využití procesoru všech modulů dohromady. Pokud je procesor moduly vytěžován nad nastavený povolený limit, je vybrán modul vytěžující procesor nejvíce.

Je-li po jedné ze dvou kontrol vybrán některý modul, dojde k jeho spuštění na záložním fyzickém stroji, na kterém běží *remote_supervizor*. Supervizor má uchovanou svou IP adresu a když se připojí *remote_supervizor* tak si uloží i IP adresu vzdáleného stroje. Spuštění vybraného modulu na vzdáleném stroji je provedeno v několika krocích. Vybraný modul je nejdříve zastaven a je mu nastaven příznak v proměnné, že bude spuštěn vzdáleně. Poté je volána funkce *graph_node_addresses_change()*, která v grafu najde uzel vybraného modulu a změní IP adresy jeho vstupních rozhraní, pokud nějaká má. Jestliže se připojuje jeho vstupní rozhraní k modulu, který běží také vzdáleně, je mu nastavena IP adresa *remote_supervizora*. Jinak je nastavena adresa supervizora. Ve stejné funkci jsou přenastaveny i IP adresy vstupních rozhraní modulů, které se připojují na vybraný modul. Těm je nastavena IP adresa *remote_supervizora* vždy.

Po přenastavení IP adres je volána funkce *send_command_to_remote()*. Pomocí této funkce supervizor posílá příkazy *remote_supervizoru*. Argumenty této funkce jsou číslo příkazu a číslo vybraného modulu s indexem do pole *running_module_t * running_modules*. Čísla příkazů jsou 1 pro spuštění nového modulu, 2 pro restartování modulu, 3 pro zastavení modulu a 4 pro opětovné spuštění již přesunutého modulu. V této fázi jsou funkci předány argumenty pro spuštění nového vybraného modulu. Funkce odešle nejdříve instanci struktury *remote_info_s*, která je zobrazena v příloze D.2. Proměnná *command_num* má hodnotu 1 pro spuštění nového modulu a proměnná *size_of_recv* má hodnotu velikosti všech dynamicky alokovaných řetězců celé struktury vybraného modulu. Tato hodnota je vypočítána funkcí *count_struct_size()*. Po odeslání této struktury je zavolána funkce, která odešle nejdříve instanci struktury *running_module_s* vybraného modulu a poté v předem určeném pořadí všechny dynamicky alokované řetězce.

Posledním krokem je restartování všech modulů, které se připojují na přesunutý modul. V grafu jsou u přesunutého modulu procházena všechna výstupní rozhraní s jejich klienty – moduly, které jsou následně restartovány. Jestliže restartovaný modul běží na lokálním stroji, je pouze zastaven a opětovným nastavením *přepínače stavu* ho servisní vlákno v zápětí znovu spustí. Běží-li restartovaný modul vzdáleně, je postupem popsaným v předešlém odstavci odeslán příkaz *remote_supervizoru*, který ho restartuje. Jediný rozdíl je v argumentu funkce *send_command_to_remote()*, které je předáno číslo příkazu 2 pro restartování modulu.

2.3.2.6 Přetížení modulu

Monitorování přetížení modulu spočívá v odeslání požadavku modulům se servisním rozhraním, přijetí statistik, aktualizaci grafu, provedení potřebných výpočtů a jejich kontrole s případným řešením přetížení. Toto je naznačeno na obrázku 1.3.

Nejdříve je odeslán požadavek ve formátu jednoho čísla typu *int* s hodnotou 1 modulům se servisním rozhraním. Podmínky pro odeslání jsou stavové informace *připojen* a *stav modulu*. Požadavek s hodnotou 1 je tedy odeslán běžícím modulům, ke kterým je supervizor připojen. Od těchto modulů je po odeslání požadavku očekávána okamžitá odpověď ve formě statistik s čítači, které jsou uloženy do struktur modulů. Poté je provedena aktualizace grafu, kterou provádí funkce *update_graph_values()*. Účelem této operace je aktualizace hodnot hran grafu, tzn. hodnoty čítačů odeslaných a přijatých zpráv. Funkce projde všechny uzly grafu a u běžících modulů se servisním rozhraním ze struktury modulu *running_module_s* aktualizuje proměnné *message_counter* všech rozhraní uzlu, který reprezentuje daný modul.

Po aktualizaci grafu jsou pomocí funkce *compute_differences()* spočteny rozdíly čítačů a odhady středních hodnot pro každé rozhraní. Tím se myslí pro každou dvojici propojených modulů, které spojuje hrana grafu. K těmto výpočtům byla vytvořena struktura *edge_statistics_s*, která je zobrazena v příloze D.7. Tuto strukturu obsahuje každé výstupní rozhraní uzlu pro všechny klienty zvlášť. Proměnná *num_periods* se rovná počtu period kontroly modulu servisním vláknem. Do proměnné *expected_value* je uložen výsledný odhad střední hodnoty, *last_period_counters_difference* se rovná rozdílu čítačů z minulé periody a *period_differences_suma* je součet všech rozdílů čítačů v každé periodě. Funkce *compute_differences()* projde všechny uzly grafu a pro každé jeho výstupní rozhraní a každého klienta daného rozhraní jsou provedeny výpočty zvlášť. Nejdříve je spočten celkový rozdíl uložených čítačů odeslaných a přijatých zpráv za celou dobu běhu modulů. Poté rozdíl za aktuální periodu rozdílem celkového rozdílu a rozdílu z minulé periody, který je uložený v proměnné *last_period_counters_difference*. Výsledek představuje ztrátovost zpráv za danou periodu a je přičten do proměnné *period_differences_suma*. Poté je spočten a uložen odhad střední hodnoty jako podíl součtu rozdílů a počtu period. Na konci je spočten odhad standardní odchylky.

Posledním krokem je kontrola výpočtů s případným řešením přetížení modulu. Pokud některý modul překročí odchylku o nastavený povolený limit, je dekomponován. Nejdříve je přetížený modul naklonován ve strukturách modulů *running_module_s*. Všechny proměnné z konfigurace má totožné a má stejná i všechna rozhraní. Dále je pro každé vstupní rozhraní přetíženého modulu přidán speciální modul *Splitter*, který má jedno vstupní rozhraní a dvě výstupní. Vstupnímu rozhraní *Splitteru* jsou nastaveny parametry vstupního rozhraní přetíženého modulu. Výstupním rozhraním *Splitteru* jsou přiděleny porty z rezervovaného rozsahu supervizorem. Tyto porty jsou poté nastaveny

vstupnímu rozhraní přetíženého modulu a jeho klonu. Dále je pro každé výstupní rozhraní přetíženého modulu přidán speciální modul *Merger*, který má dvě vstupní rozhraní a jedno výstupní rozhraní. Výstupnímu rozhraní *Mergeru* jsou nastaveny parametry výstupního rozhraní přetíženého modulu a výstupnímu rozhraní klonu a přetíženého modulu jsou přiděleny nové porty, které jsou poté nastaveny vstupním rozhraním *Mergeru*. Díky modulu *Splitter* budou data rozdělena mezi dva totožné moduly a modul *Merger* výstup od obou zase sloučí dohromady. V příloze E obrázky E.5 a E.6 ukazují stav zapojení před dekompozicí modulu a po dekompozici modulu.

2.4 Dokumentace programu

Nápověda programu

Pro uživatele, kteří se supervizorem budou pracovat poprvé byla do programu zakomponována nápověda, popisující dostupné operace a příklad vstupní konfigurace modulů ve formátu XML. Tato nápověda je vyvolána při spuštění supervizora s přepínačem -h. Supervizor jen vypíše nápovědu a ukončí se. V tomto případě není parametr -f s konfiguračním souborem povinný . Příklad spuštění *./supervisor -h* nebo *supervisor -h* pokud je nainstalovaný. Výpis nápovědy je obsažen v příloze E na obrázku E.4.

Dokumentace kódu

Součástí zdrojových kódů jsou pro lepší orientaci a porozumění komentáře. Okomentované jsou všechny funkce s návratovou hodnotou a argumenty, struktury s jejich proměnnými, makra a globální proměnné. Ze zdrojových kódů je díky tomu možné vygenerovat dokumentaci pro vývojáře.

Kapitola 3

Testování

3.1 Testování programu při vývoji

Během celého průběhu vývoje jsem veškeré funkcionality testoval na dostupných modulech systému Nemea. Implementované funkce a části supervizora jsem testoval a přizpůsoboval modulům. První a pro systém zcela zásadní část byla načtení a parsování XML konfigurace a poté předání správných parametrů při spuštění modulu. Parametry jsou tvořeny pro libovolný modul s dopředu neznámým počtem a typem rozhraní a parametrů. Testování na různých modulech bylo tedy nezbytné.

Z testování na modulech vzešlo během vývoje mnoho užitečných poznatků, které jsem později zapracoval do programu. Příkladem může být funkce, která před spuštěním modulu projde zapojení modulů v grafové struktuře programu a hledá duplicitní porty. V případě nalezení modulu se stejným portem výstupního rozhraní je nahlášena chyba uživateli a modul není spuštěn. Důvodem tohoto opatření je fakt, že výstupní rozhraní funguje jako server čili na přiděleném portu poslouchá a další modul při stejné operaci neuspěje, protože daný port je již obsazený. Následně se modul ukončí. Problém nastal s funkcí supervizora monitorování stavu modulu, kdy opakovaně znovu spouštěl modul, který nemohl poslouchat na přiděleném portu v domnění, že u něj došlo k chybě, tudíž je třeba ho znovu spustit. Modul ale opakovaně neuspěl a dokola se ukončoval.

K odhalení možných chyb při běhu programu nebo neuvolněné paměti jsem průběžně používal program valgrind. Při hledání chyb jsem také mnohokrát použil program gdb. Všechny nalezené problémy byly odstraněny.

3.2 Nasazení supervizora a jeho testování

Supervizor je prozatím testován na jednom ze serverů CESNETu. Ostré nasazení spolu s vydáním první stabilní verze supervizora je plánováno v květnu 2014 v dalším vydaném balíku systému Nemea.

Vedoucí práce provádí testování na serveru, kde spustil supervizora spolu s konfigurací 18 modulů. Jedná se o entropy moduly, které provádí výpočet entropií z informací o síťových tocích na každé z 9 sond. Ke každému entropy modulu je připojen modul logger, který zaznamenává výsledky výpočtů. Tuto konfiguraci supervizorem konfiguruje a monitoruje.

Po celou dobu vývoje byla práce verzována verzovacím nástrojem GIT. K repozitáři měli přístup zadavatelé práce i vedoucí práce a mohli tak sledovat postup vývoje.

3.3 Testování použitelnosti

Pravidelnými konzultacemi se zaměstnanci CESNETu a budoucími uživateli supervizora jsem průběžně sbíral poznámky k jednotlivým funkcionalitám programu. Po prezentacích funkčních verzí programu jsem zároveň sbíral připomínky k ovládání a také možné problémy supervizora. Tyto problémy jsem vždy následně vyřešil.

3.3.1 Načtení konfiguračního souboru

Pro načtení konfigurace ze souboru do vnitřních struktur programu jsem nejdříve používal statické řetězce a proměnné. Už při prvním testování jsme ale narazili na problém s velikostmi řetězců v konfiguračním souboru, se kterými jsem nepočítal a program kvůli tomu tedy padal a byl nepoužitelný.

Řešení problému: všechny proměnné načítané z konfiguračního souboru alokovat dynamicky při parsování. Vyřeší to jednak použitelnost konfiguračního souboru a také paměťovou náročnost alokací, protože díky tomu není alokováno více paměti než je potřeba.

3.3.2 Výstupy programu

Při první prezentaci fungující verze programu se jedna připomínka týkala výstupu programu. Do terminálu se uživateli na standardní výstup vypisují dostupné operace a po vybrání jedné z nich se nabídka zobrazí znovu, aby uživatel seznam neztratil. Spolu s nabídkou se ale vypisovalo do terminálu vše ostatní jako statistiky od modulů, události modulů (spuštění, restartování apod.) a to bylo pro uživatele velice nepřehledné.

Řešení problému: jedním z možných návrhů řešení bylo udělat volitelnou podrobnost výpisů supervizora. Já jsem ale navrhl a provedl rozdělení výpisů podle kategorie do různých souborů a na standardním výstupu v terminálu se uživateli vypisuje tedy jen nabídka operací spolu s upřesňujícími dotazy při jejich provádění. Výstup programu byl popsán v sekci 1.6.4.

Pokud supervizor běžel na pozadí systému delší dobu, vznikl další problém a to velikost výstupních souborů. Zejména soubor se statistikami by při spuštění desítek modulů po delší době zabíral mnoho místa.

Řešení problému: navrhl jsem techniku rotování dvou logovacích souborů. Pokud jeden soubor dosáhne určité velikosti, zapisuje se do druhého. Když se zaplní i druhý soubor, přepíše se opět ten první. Tímto řešením se šetří místo na disku a zároveň jsou statistiky za určitou dobu stále k dispozici.

3.3.3 Znovu spouštění modulů

První testy ukázaly, že monitorování stavu modulu a následné znovu spuštění neběžících povolených modulů je nepoužitelné v případě, že modul má nějaký problém, který znovu spuštění nevyřeší. V takovém případě by supervizor znovu spouštěl modul do nekonečna.

Řešení problému: řešení spočívá ve stanovení maximálního počtu znovu spuštění za určitou dobu. Po konzultacích jsem stanovil maximální počet znovu spuštění na 3 za jednu minutu. Tyto hodnoty jsou přípustné. Dojde-li k jejich překročení, modul je označen jako nefunkční a není dále znovu spouštěn.

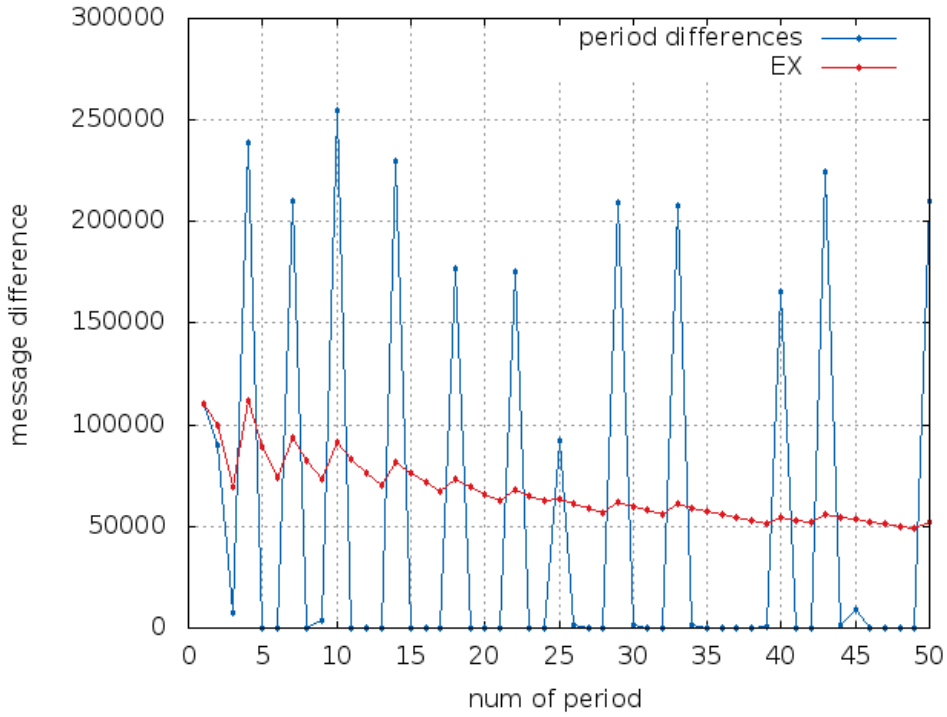
3.4 Simulace zatížení modulu

Supervizor monitoruje tok zpráv v systému. Podle ztrátovosti zpráv poté detekuje možné přetížení modulu, které bylo vysvětleno v sekci 1.7.2. Abych otestoval chování systému, provedl jsem simulaci pravidelného zatížení modulu. Modul X odesílal neustále zprávy modulu Y, který simuloval pravidelný výpočet. Simulovaný výpočet představovalo uspání modulu na určený počet milisekund, které následovalo po přijetí určeného počtu zpráv. Toto se periodicky opakovalo.

Ve chvíli, kdy modul Y prováděl simulovaný výpočet a nepřijímal proto zprávy, modul X je zahazoval. Ztrátovost zpráv v tu chvíli rapidně narůstá.

Simulaci jsem provedl pro délku pravidelného výpočtu 150 ms a 300 ms. Druhým parametrem simulace byla četnost výpočtů a tu jsem stanovil provedením výpočtu po přijetí 5 milionů zpráv a 10 milionů zpráv. Provedl jsem tedy 4 různé simulace. Na obrázku 3.1 je znázorněn výsledek simulace pro délku výpočtu 150 ms po přijetí 5 milionů zpráv. Na ose X je vyneseno číslo periody kontroly supervizora. Na ose Y je vynesen rozdíl zpráv odeslaných a přijatých modulů X a Y v dané periodě kontroly, což je ztrátovost zpráv. Modrá křivka znázorňuje ztrátovost jednotlivých period a červená křivka znázorňuje odhad střední hodnoty ztrátovosti. Ten se v této simulaci po 50 periodách kontroly supervizora blížil 50 000 zpráv za periodu.

V příloze E jsou na obrázku E.1, E.2 a E.3 zobrazeny i zbylé 3 simulace.



Obrázek 3.1: Graf zobrazuje výsledek simulace zatížení modulu při délce výpočtu 150 ms pravidelně po přijetí 5 milionů zpráv. Modrá křivka zobrazuje ztrátovost zpráv v jednotlivých periodách kontroly supervizorem a červená odhad střední hodnoty ztrátovosti, která se po 50 periodách blíží k 50 000 zpráv za periodu.

Závěr

Vývoj supervizora pro mne byl velkým přínosem. Získal jsem mnoho nových poznatků o unixových systémech a standardní knihovně jazyka C. Mohl jsem také uplatnit znalosti a dovednosti získané po dobu studia. Přínosem je i vývoj daleko rozsáhlejšího vícevláknového programu než jsem byl doposud zvyklý. Tento programu představují tři spustitelné části, které běží ve stejnou dobu a komunikují spolu.

Důležitou zkušeností pro mě je i spolupráce s CESNETem na takto rozsáhlém projektu. Celou dobu jsem se účastnil schůzí, prezentací a konzultací. Díky zpětné vazbě zaměstnanců a budoucích uživatelů supervizora jsem vyvíjel program podle požadavků a připomínek.

Shrnutí průběhu práce, vlastního přínosu a výsledků práce

Nejdříve jsem musel podrobně nastudovat fungování systému Nemea a také knihovny libtrap, do které jsem později implementoval část servisního rozhraní modulů.

Před začátkem implementace jsem na základě konzultací se zaměstnanci CESNETu sbíral a analyzoval požadavky na funkcionalitu supervizora. Konzultace pokračovaly po celou dobu vývoje programu.

Prezentace programu s následným testováním přinesly mnoho přínosných poznatků a odhalení chyb. Po jejich odstranění je supervizor pro uživatele lépe pochopitelný a snáze se ovládá.

Má práce do systému Nemea přinesla dosud neexistující konfigurační a monitorovací nástroj modulů. Tento nástroj (supervizor) umožňuje spravovat uživatelem vytvořenou konfiguraci modulů pomocí operací spuštění a vypnutí modulu, zobrazení stavu modulů, spuštění dodatečné konfigurace či grafického zobrazení zapojených modulů. Hlavním úkolem supervizora je sledování stavu modulů a zatížení systému. Na základě sbíraných informací je supervi-

zor schopen rozhodnout, zda je výhodné rozložit zátěž jednoho modulu mezi více modulů nebo dokonce i na více fyzických strojů.

Všechny požadavky budoucích uživatelů i zadání této práce byly splněny. Supervizor je hotový a připravený k nasazení. Jako přínos chci zmínit i klienta k supervizoru, který není součástí zadání a původně nebyl ani v plánu. Díky němu je možné program ovládat i když běží jako služba operačního systému.

Budoucí rozšiřování

Výsledkem mé práce je funkční back-end supervizor. V dalších měsících je plánováno jeho propojení s front-endem v podobě už existujícího a v CESNETu používaného web-GUI. Toto GUI komunikuje se síťovými prvky přes protokol NETCONF. Jedná se o protokol, který poskytuje prostředky k získávání stavových informací, nastavení, manipulaci a mazání konfigurace síťových zařízení. Využívá k tomu kódování založené na XML, které je použito také pro zprávy protokolu. Operace protokolu jsou pak realizovány jako vzdálené volání procedur (remote procedure calls - RPCs).

Rozšíření supervizora bude spočívat ve vytvoření několika API funkcí, které bude protokol NETCONF využívat.

Literatura

[1] Systém Nemea. [Citováno 2014-03-16]. Dostupné z: https://www.liberouter.org/technologies/nemea/

[2] The XML C parser and toolkit of Gnome. [Citováno 2014-03-16]. Dostupné z: http://www.xmlsoft.org/

[3] Bartoš, V.; Žádník, M.; Čejka, T.: Nemea: Framework for streamwise analysis of network traffic. Technická zpráva, CESNET, a.l.e., 2013. Dostupné z: http://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf

[4] Blažek, R.: Bodové odhady parametrů [online prezentace]. 2013, [Citováno 2014-02-25]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-PST/_media/lectures/9_pred_pst-v2.pdf

[5] Kolář, J.: Planarita a toky v sítích [online prezentace]. 2013, [Citováno 2014-02-25]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-GRA/_media/lectures/gra-predn-09cb.pdf

[6] Kolář, J.: Souvislost, Orientované grafy [online prezentace]. 2013, [Citováno 2014-02-25]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-GRA/_media/lectures/gra-predn-02cb.pdf

[7] Čejka, T.: Systém pro detekci anomálií v počítačových sítích. Technická zpráva, CESNET, a.l.e., 2013.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

API Application programming interface

SSH Secure Shell

XML Extensible markup language

PNG Portable Network Graphics

TCP Transmission Control Protocol

CPU Central processing unit

PID Process identifier

IP Internet Protocol

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `nemea-supervisor-1.0.0.tar.gz` ..... instalační balík
  - `implementace modulu` ..... zdrojové kódy implementace
- `doc` ..... dokumentace modulu
- `text` ..... text práce
  - `BP_Švepeš_Marek_2014.pdf` ..... text práce ve formátu PDF
  - `BP_Švepeš_Marek_2014.tex` .. zdrojová forma práce ve formátu LaTeX

Příloha C

Instalační manuál

Instalace modulu supervizor byla otestována na Ubuntu Linux distribuci. Pro správný chod supervizora je potřebná knihovna libxml2 [2], program dot(1) a program display(1). Po spuštění *configure* skriptu jsou provedeny kontroly přítomnosti těchto prvků. V případě, že chybí program dot nebo display, *configure* zobrazí upozornění. Knihovna libxml2 je ale nezbytná pro fungování supervizora a instalace bez ní nebude dokončena.

C.1 Instalace modulu

Pro nainstalování modulu je potřeba stáhnout archiv *nemea-supervisor-1.0.0.tar.gz* z přiloženého CD a v adresáři s archivem provést tyto příkazy:

```
$ tar -zxvf nemea-supervisor-1.0.0.tar.gz -C [cílové umístění]
$ cd [cílové umístění]
$ ./configure
$ make install
```

U skriptu *configure* je možné nastavit cestu k adresáři, kam se modul nainstaluje takto:

```
$ configure --prefix=[cílové umístění]
```

C.2 Spuštění modulu

Modul supervizor tvoří 3 spustitelné soubory. Soubor *supervisor* spustí hlavní funkční část, soubor *supervisor_cli* klientskou část pro ovládání systémového procesu na pozadí a soubor *supervisor_remote* se spouští na záložním fyzickém stroji pro případné rozložení zátěže systému.

Při prvním spuštění jistě pomůže nápověda, která se zobrazí spuštěním příkazu *supervisor -h*. Spuštění modulu se potom provede příkazem

$ supervisor -f konfigurační_soubor.xml [--daemon].

Příloha D

Použité datové struktury

```
typedef struct running_module_s {
  int                   module_cloned;
  int                   module_served_by_service_thread;
  int                   module_ifces_array_size;
  int                   module_running;
  char *                module_params;
  int                   module_enabled;
  int                   module_ifces_cnt;
  int                   module_num_out_ifc;
  int                   module_num_in_ifc;
  int *                 module_counters_array;
  int                   module_service_ifc_isconnected;
  int                   module_has_service_ifc;
  int                   module_service_sd;
  char *                module_name;
  int                   module_status;
  int                   module_restart_cnt;
  pid_t                 module_pid;
  char *                module_path;
  interface_t *         module_ifces;
  int                   last_cpu_usage_kernel_mode;
  int                   last_cpu_usage_user_mode;
  int                   percent_cpu_usage_kernel_mode;
  int                   percent_cpu_usage_user_mode;
  int                   num_periods_overload;
  int                   remote_module;
  int                   module_number;
} running_module_t;
```

Výpis D.1: .

```
typedef struct remote_info_s {
  int            command_num;
  int            size_of_recv;
} remote_info_t;
```

Výpis D.2: .

```
typedef struct interface {
  char *        ifc_note;
  char *        ifc_type;
  char *        ifc_params;
  char *        ifc_direction;
} interface_t;
```

Výpis D.3: .

```
typedef struct graph_node_input_interface_s {
  interface_t *                          ifc_struct;
  graph_node_t *                         parent_node;
  int                                    message_counter;
  int                                    node_interface_port;
  graph_node_output_interface_t *        node_interface_output_ifc;
} graph_node_input_interface_t;
```

Výpis D.4: .

```
typedef struct graph_node_output_interface_s {
  interface_t *                          ifc_struct;
  graph_node_t *                         parent_node;
  int                                    message_counter;
  int                                    node_children_counter;
  graph_node_input_interface_t **        node_children;
  int                                    node_interface_port;
  edge_statistics_t *                    statistics;
} graph_node_output_interface_t;
```

Výpis D.5: .

```
typedef struct graph_node_s {
  void *                                 module_data;
  int                                    module_number;
  graph_node_t *                         next_node;
  graph_node_input_interface_t *         node_input_interfaces;
  graph_node_output_interface_t *        node_output_interfaces;
  int                                    num_node_input_interfaces;
  int                                    num_node_output_interfaces;
} graph_node_t;
```

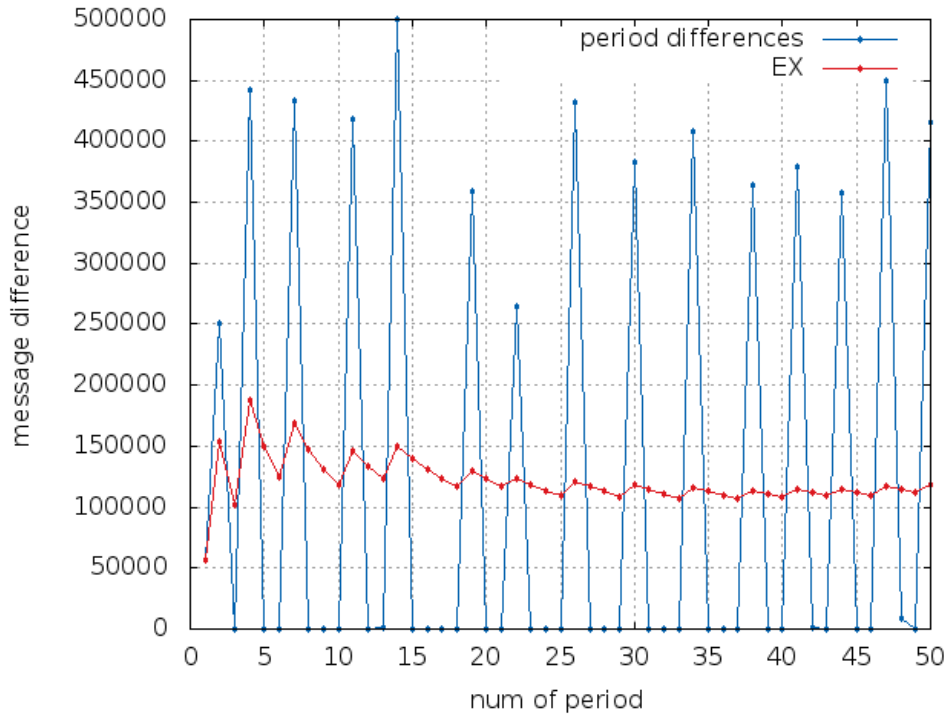
Výpis D.6: .

```
typedef struct edge_statistics_s {
  int           last_period_counters_difference;
  int           num_periods;
  int           expected_value;
  int           period_differences_suma;
} edge_statistics_t;
```

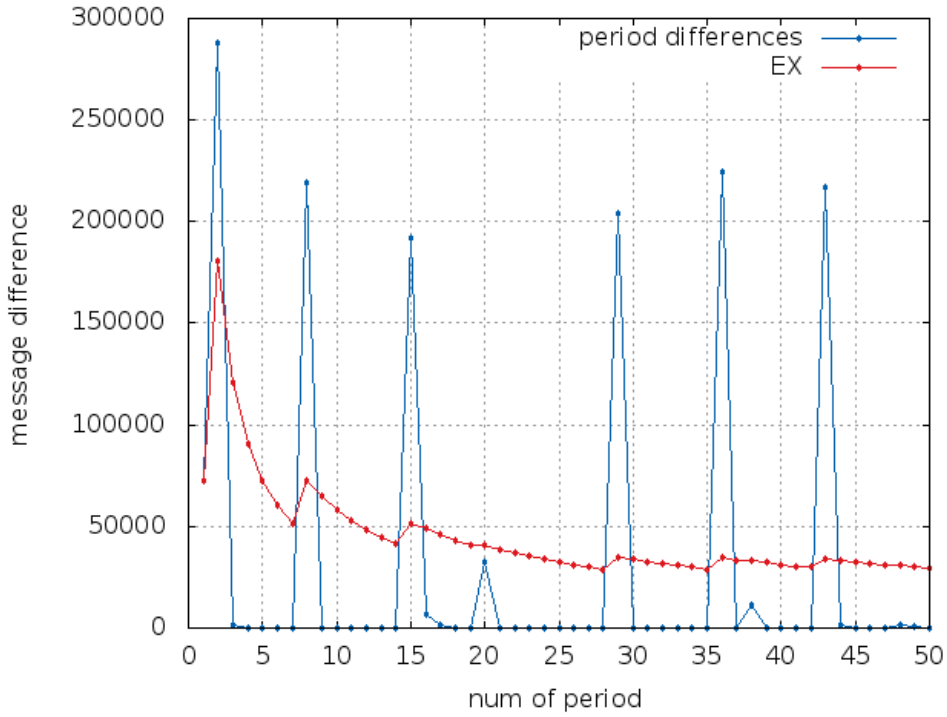
Výpis D.7: .

Příloha E

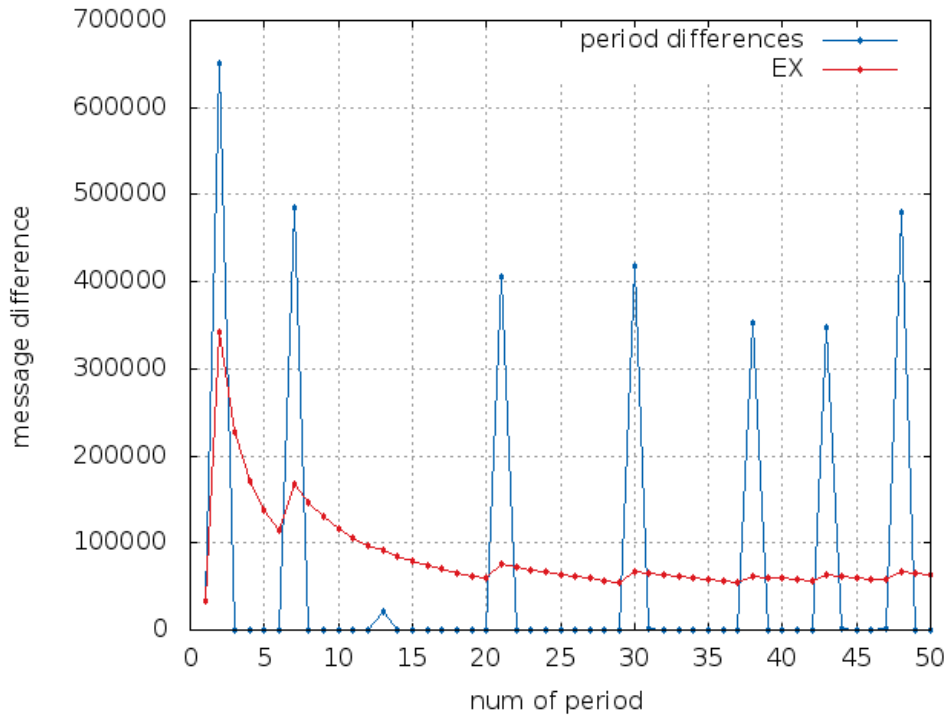
Obrázkové přílohy



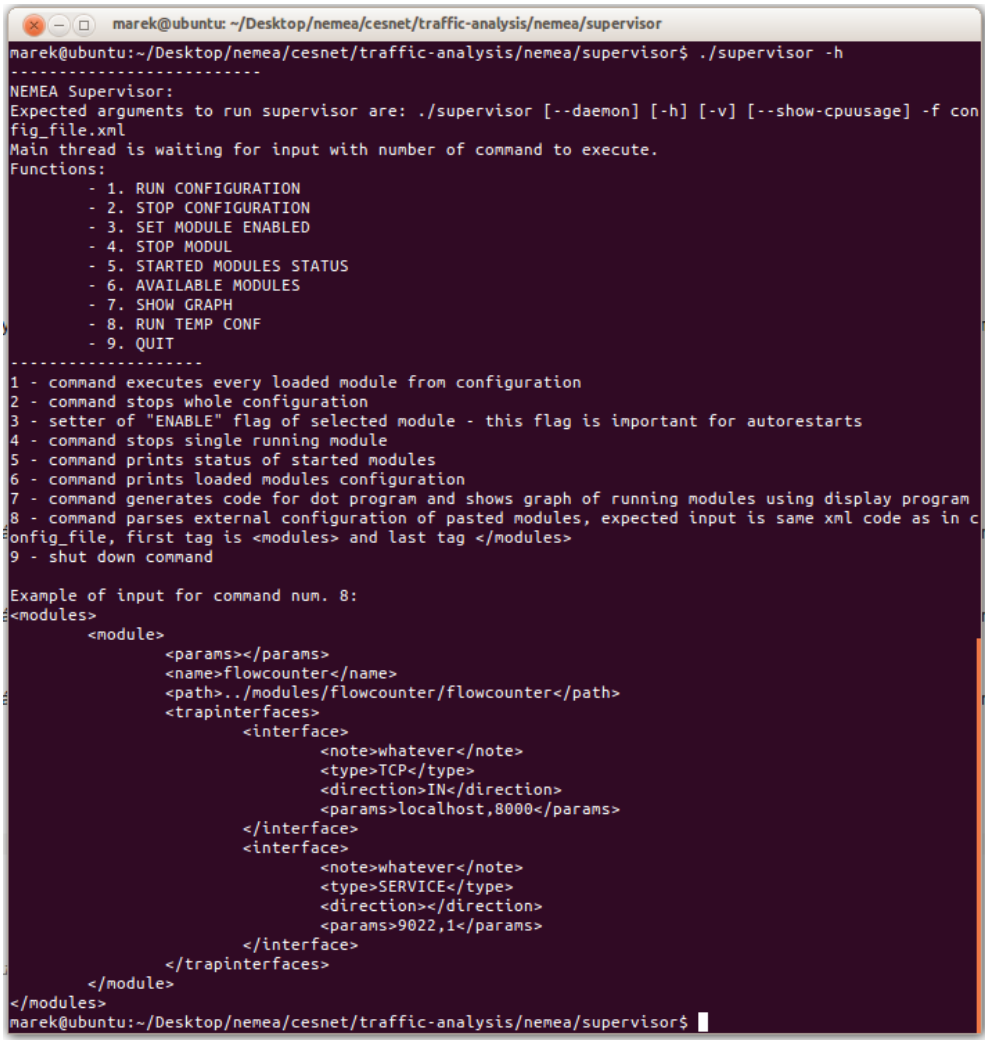
Obrázek E.1: Graf zobrazuje výsledek simulace zatížení modulu při délce výpočtu 300 ms pravidelně po přijetí 5 milionů zpráv. Modrá křivka zobrazuje ztrátovost zpráv v jednotlivých periodách kontroly supervizorem a červená odhad střední hodnoty ztrátovosti, která se po 50 periodách blíží k 120 000 zpráv za periodu.



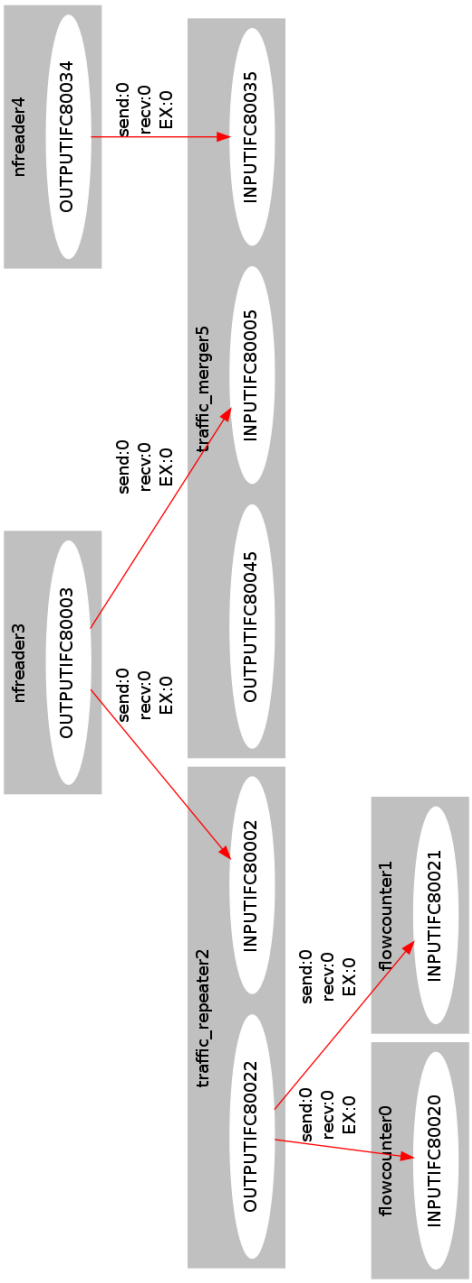
Obrázek E.2: Graf zobrazuje výsledek simulace zatížení modulu při délce výpočtu 150 ms pravidelně po přijetí 10 milionů zpráv. Modrá křivka zobrazuje ztrátovost zpráv v jednotlivých periodách kontroly supervizorem a červená odhad střední hodnoty ztrátovosti, která se po 50 periodách blíží k 27 000 zpráv za periodu.



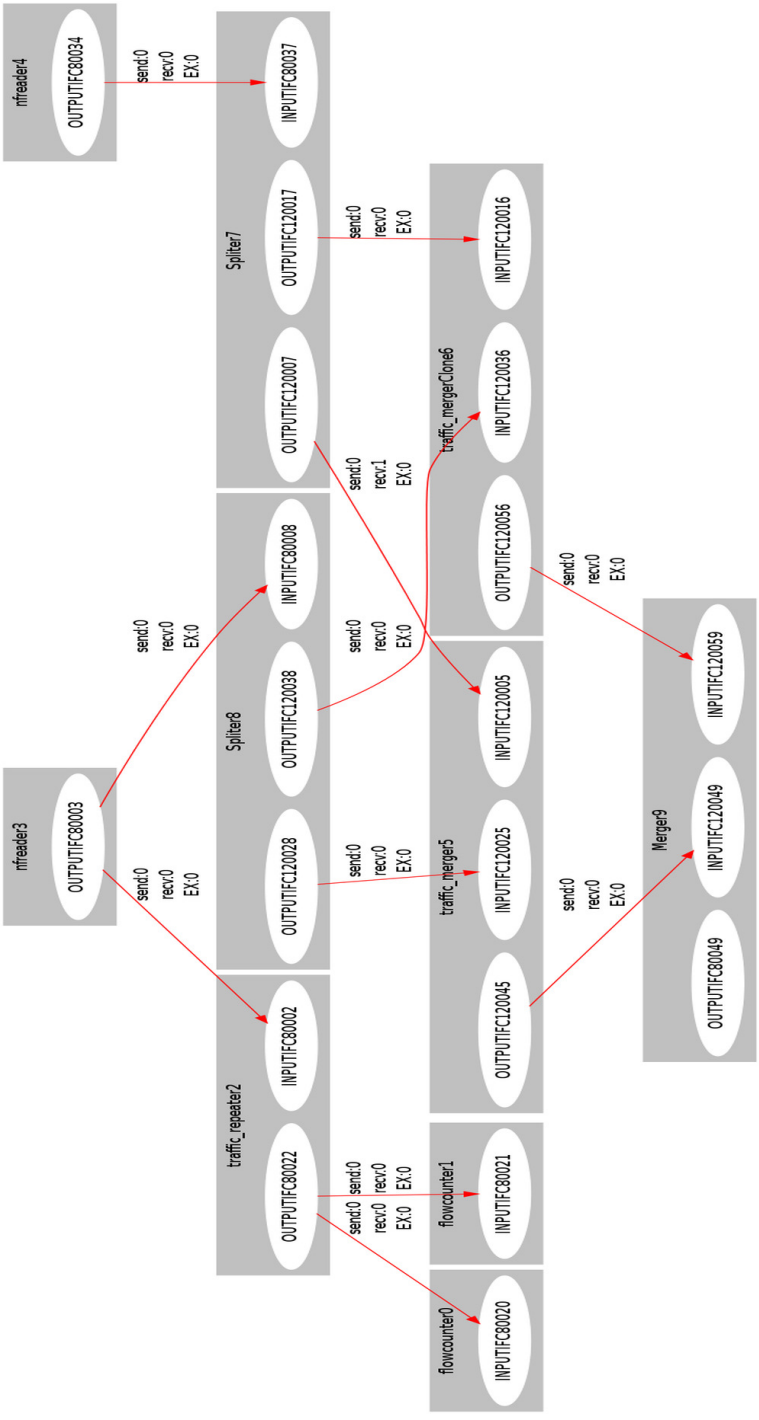
Obrázek E.3: Graf zobrazuje výsledek simulace zatížení modulu při délce výpočtu 300 ms pravidelně po přijetí 10 milionů zpráv. Modrá křivka zobrazuje ztrátovost zpráv v jednotlivých periodách kontroly supervizorem a červená odhad střední hodnoty ztrátovosti, která se po 50 periodách blíží k 60 000 zpráv za periodu.



Obrázek E.4: Ukázka výstupu uživatelské nápovědy programu, která je v terminálu zobrazena po spuštění programu s přepínačem *-h*.



Obrázek E.5: Ukázka zapojení modulů před dekompozicí přetíženého modulu *traffic_merger5*. Zapojení po dekompozici ukazuje obrázek E.6.



Obrázek E.6: Ukázka zapojení modulů po dekompozici přetíženého modulu *traffic_merger5*. Data jsou rozdělena mezi dva totožné moduly pomocí modulu *Splitter* a po zpracování sloučena modulem *Merger*.