VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MONITOROVÁNÍ APLIKAČNÍ VÝKONNOSTI HTTP

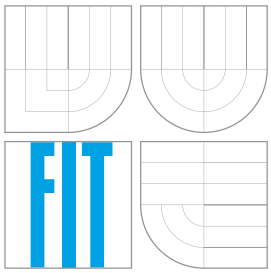
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN KNAPIK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MONITOROVÁNÍ APLIKAČNÍ VÝKONNOSTI HTTP

HTTP APPLICATION PERFORMANCE MONITORING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN KNAPIK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV BARTOŠ

BRNO 2015

Abstrakt

Cílem této bakalářské práce bylo vytvořit řešení pro monitorování a analýzu síťové výkonnosti HTTP serverů s využitím frameworku Nemea a NetFlow záznamů. Pro tento účel jsem vytvořil modul ve frameworku Nemea, který filtruje, rozebírá a ukládá NetFlow záznamy obohacené o informace z HTTP pluginu ve flow exportéru. Následné bylo potřebné vytvořit webové rozhraní založené na frameworku Django, pro zobrazení různych štatistík, které může užívatel využít na zjištení problému s monitorovanými servery. Výsledkom mé práce je produkt, který demonstruje možnost využití systému Nemea na pasívní monitorování HTTP servrů.

Abstract

Goal of this bachelor thesis was to create solution for monitoring and analysis of network performance of HTTP server using Nemea framework and NetFlow data. For this purpose, I've created Nemea module for filtering, parsing and saving NetFlow data enhanced by informations gained from HTTP plugin on exporter. For analysis and user interface, webpage based on Django framework was created, used for displaying statistics that are useful for users in order to reveal problems with monitored servers. Result of my work is product, which is demonstrating possibility of using of Nemea system for passive monitoring of HTTP servers.

Klíčová slova

pasívní monitorování, Nemea, NetFlow, měření výkonnosti HTTP

Keywords

passive monitoring, Nemea, NetFlow, HTTP performance monitoring

Citace

Martin Knapik: Monitorování aplikační výkonnosti HTTP, bakalářská práce, Brno, FIT VUT v Brně, 2015

Monitorování aplikační výkonnosti HTTP

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Václava Bartoša. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Martin Knapik
20. května 2015

Poděkování

Chcel by som poďakovať svojmu vedúcemu, pánu Bartošovi, za neustálu pomoc a podporu a takisto Jakubovi Budiskému za poskytnutie pluginu do exportéru.

Obsah

Kapitola 1

Úvod

Každá sekunda načítania stránky môže spôsobiť až 7 percentný pokles predajov, takéto vyhlásenie zaznelo na konferencií Strange Loops[5]. Nieje to však jediné vyhlásenie týkajúce sa vzťahu medzi rýchlosťou načítania stránky a ziskov zo stránky, prípadne jej návštevnosti. Pravdou je že v dnešnej dobe užívateľov stránok, viac než dizajn a funkcionalita stránky zaujíma doba ktorá prebehne medzi tým ako sa rozhodnú navštíviť stránku a zobrazením stránky. Dokonca sa môže stať že pokiaľ je táto doba privysoká, rozhodne sa užívateľ využiť konkurenčnú stránku. Nesmieme samozrejme zabúdať to, že jeden z najpoužívanejších vyhľadávačov stránok Google, využíva rýchlosť načítania stránky ako jeden z parametrov podľa ktorých rozhoduje na ktorom mieste vyhľadávania našu stránku zobrazí.

Problémom v dnešnej dobe je často to, že sa administrátor stránky mnohokrát ani nedozvie že je problém s jeho stránkou, až pokiaľ nezačnú užívatelia odchádzať zo stránky, čo je zvyčajne už neskoro. Napomôcť s týmto problémom môže monitorovanie výkonu HTTP stránok, ktoré umožňuje zobrazovať problémy so stránkou ešte pred ich výskytom, najneskôr však pri ich prvom výskyte. Existuje mnoho nástrojov ktoré slúžia na rôzne druhy monitorovanie stránok, v tejto bakalárke sa však pokúsim načrtnúť riešenie monitorovania s využitím frameworku Nemea vyvíjaného skupinou Liberouter v organizácií CESNET. Prvé dve časti tejto práce budú hovoriť o veciach ktoré bolo potrebné naštudovať v súvislosti s monitorovaním a s nástrojmi ktoré boli použité k vytvoreniu konečného produku. Nasledovať bude návrh riešenia a napokon samotná implementácie tohto riešenia.

Kapitola 2

Monitorovanie HTTP výkonnsti

Monitorovanie webu slúži k testovaniu a ukladaniu stavu a výkonnosti jedného alebo viacerých webových stránok[11]. Výsledkom by malo byť odhalenie slabín v dostupnosti a výkonnosti nášho webu ktoré by mohli pokaziť užívateľov dojem zo stránky (end-user experience).

Túto kapitolu začnem obecnými pravdlami pri monitorovaní webu, pokračovať budem základným popisom HTTP protokolu a jeho životného cyklu, keďže tvorí základný protokol pri práci s web stránkami. Pridám ešte niekoľko pojmov z oblasti HTTP ktoré je potrebné poznať pre pochopenie niektorých problémov ktoré sa vyskytli pri tvorbe výsledného softvéru.

2.1 Monitorovanie webu

Účel monitorovania webu je testovanie a overovanie že užívatelia budú schopní pracovať s webovou stránkou alebo aplikáciou podľa očakávani. Tento proces sa často využíva v spoločnostiach prevádzkujúcích hosting web stránok alebo web aplikácií, aby zabezpečili beh, výkon a funkcionalitu týchto služieb.

Význam monitorovania webu spočíva v tom, zabezpečiť dostupnosť a primeranú rýchlosť stránky pre užívateľov ktorý na túto stránku spoliehajú, či už z pracovných dôvodov alebo iných. Monitorovanie ma za úlohu odhaliť nedostatky ktoré sa môžu týkať hardvéru na ktorom stránka beží, pripojenia alebo stránky samotnej.
Monitorovanie rozdeľujeme do dvoch hlavných kategórií.

Aktívne monitorovanie je založené na opakovanom púšťaní nahraných akcií niekoľko krát denne. Tieto akcie simulujú užívateľovo správanie na stránke. Tento prístup umožňuje monitorovať a vyhodnocovať kritické prípady aj bez toho aby reálne nastali. Týmto umožňujú administrátorovi stránky odhaliť chyby aj predtým ako nastanú pri behu stránky.

Pasívne monitorovanie monitoruje skutočné akcie zo sieťovej prevádzky vytvorené užívateľom pri interakcií so stránkou, bez toho aby do nej akýmkoľvek spôsobom zasahovalo. Nevýhodou tejto techniky je to, že pomocou nej je možné odhaliť chyby až potom čo nastanú. Na druhej strane je však zavedenie tohto monitorovania na funkčný web omnoho rýchlejšie.

Väčšinou sa však využíva kombinácia oboch techník, kde aktívne monitorovanie slúži na otestovanie stránky predtým ako sa stránka dostane k užívateľovi, najmä na odhalenie kritických

miest v ktorých by mohol vzniknúť problém. Potom čo sa stránka dostane k užívateľom nahradí sa aktívne monitorovanie pasívnym, ktorého úlohou je odhaliť problémy ktoré nastali, aby sme mohli ľahšie odhaliť zdroj chyby a odstrániť ho. Aktívne monitorovanie sa v tomto okamihu už nevyužíva, prípade iba vtedy keď je prevádzka na serveri menšia, keďže táto technika vytvára zbytočnú záťaž na server[6]. Táto práca sa bude ďalej zaoberať výhradne pasívným monitorovaním.

2.2 Metriky pre meranie výkonnosti

Keďže užívateľov dojem zo stránky je z veľkej časti subjektívny, čiže nemerateľný pomocou automatických nástrojov, bolo potrebné zaviesť merateľné metriky z ktorých by bolo možné približne určiť užívateľovú spokojnosť so stránkou. Je však potrebné mať na pamäti že výsledky meraní su ovplyvnené nielen serverom, ale mnohokrát aj klientom ktorý môže mať napríklad problémy s pripojením a takisto nesmieme zabúdať na to že proces komunikácie servera s klientom je vo veľkej miere ovplyvnený typom prehliadača aký klient vyžíva. Jedná sa o tieto metriky:

Doba odozvy reprezentuje čas ktorý trvá od zahájenia komunikácie klientom až po jej ukončenie, dlhšia doba odozvy môže znamenať že server má pomalé pripojenie k internetu a preto trvá každé spojenie dlhšie než je očakávané, ale takisto môže znamenať že si klient vyžiadal väčší súbor.

Oneskorenie označuje čas medzi odoslaním prvého požiadavku od klienta a zaslatím odpovede servera, v prípade že je tento čas dlhší než by sme očakávali je možné že server nestíha spracúvať požiadavky takou rýchlosťou akou sú mu zasielané.

Počet užívateľov je meraná ako počet unikátných adries ktoré sa pripojili k serveru za určité časové obdobie. Toto môžme okrem iného využit na rátanie počtu užívateľov na serveri za určité rozsiahlejšie obdobie (deň, mesiac), čím môžme odhaliť časy kedy nieje pravdepodobná vysoká prevádzka na serveri. Túto informáciu môžme využiť v prípade že potrebujeme na servery vykonať údržbu, alebo zálohovať dáta čo vytvorí veľkú záťaž na server.

Počet požiadavkov , túto metriku je možné merať najjednoduchšie ako počet odoslaných požiadavkov serveru za určite obdobie. Slúži nám skôr ako pomocná premenná pri zisťovaní dôvodov pre vysokú dobu odozvy a oneskorenia, prípadne môžme znova použiť na zisťovanie predpokladanej záťaže na server.

Chyby označujú počet chýb zaslaných klientovi, pričom od druhu chyby sa môže buď jednať o vypršanie spojenia z dôvodu veľmi pomalého pripojenia alebo nenájdenej stránky z dôvodu nefunkčného odkazu.

Širka pásma je veľkosť pásma ktorú sme využili na odoslanie odpovede, príliš vysoká môže znamenať že zasielame zbytočné množstvá dát (napr. nesprávny formát obrázkov).

Z týchto údajov si následne môžme odvodiť systémovú výkonnosť servera a užívateľom vnímanú výkonnosť servera v určitých situáciách.
Systémová výkonnosť servera nám môže povedať ako sa zmení schopnosť servera odpovedať na požiadavky v prípade že množstvo požiadavok zvýši niekoľkonásobne, na druhej strane užívateľom vnímaná výkonnosť nám poukáže na to čo je mnohokrát dôležitejšie, a to ako sa zmení užívateľov dojem zo stránky v takýchto prípadoch[1].

2.3 HTTP

HTTP (Hypertext Transfer Protocol) je protokol na aplikačnej vrste, ktorého najčastejšie využitie je distribúcia www stránok na internete. Nemá zmysel sa v tejto časti zaoberať všetkými detailami tohto protokolu, keďže aj samotná práca využíva iba niektoré ku ktorým má kvôli svojím obmedzeniam prístup. HTTP vo všeobecnosti pracuje na transportnom protokole TCP, pričom servre prijímajú dáta na porte 80 (príp. 8080).
Životný cyklus takejto komunikácie je následovný:

1. **Handshake**
   Každá komunikácia začína nadviazaním spojenia, pričom klient začína komunikáciu, nasleduje inicializácia spojenia.

2. **Request**
   Potom čo prebehne inicializácia zašle klient serveru požiadavok na obsah stránky, klientom rozumieme väčšinou prehliadač web stránok.

3. **Response**

   (a) Server odošle potvrdenie o prijatí požiadavku.

   (b) Vygeneruje odpoveď a odošle ju klientovi.

4. **Acknowledgment**
   Klient po prijatí odpoveďe zašle potvrdenie o jej prijatí, v tomto okamihu sa zvyčajne zobrazí obsah stránky užívateľovi.

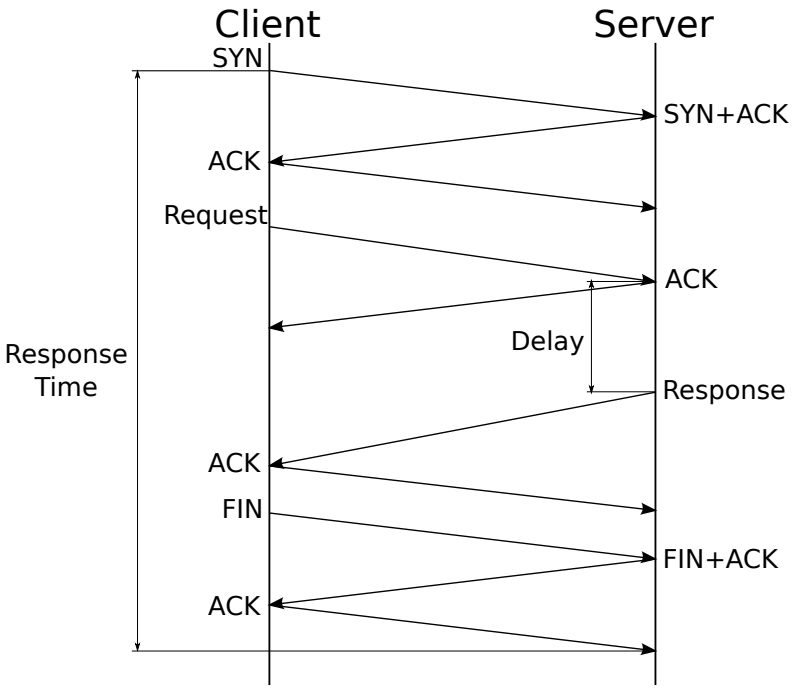
5. **Close**
   Klient zasiela požiadavok na ukončenie spojenia, po výmene správ je komunikácia ukončená

Grafická reprezentácia tohto procesu je znázornená na obrázku 2.1. Vo svetle tohto postupu si môžme odvodiť že pri monitorovaní HTTP toku bude doba odozvy reprezentovať čas medzi handshakom (1) až po zaslatie odpovede a ukončenie spojenia (5). Oneskorenie bude v tomto prípade označovať čas medzi odoslatím požiadavku klientom (2) po odoslanie odpovede klientovi (3b). Ďalšia metrika ktorú môžme odvodiť je čas medzi odoslaním potvrdenia (3a) o prijatí požiadavku po odoslanie odpovede (3b), toto reprezentuje čistý čas generovania odpovede.

Trvalé spojenie

Od HTTP 1.1 bola pridaná podpora *trvalého spojenia (keep-alive connection)* ktoré umožňuje po prijatí odpovede od servera toto spojenie ďalej využívať miesto toho aby sme ho po prijatí odpovede zatvorili a kvôli ďalšiemu požiadavku vytvárali nové. Takýto postup šetrí výkon procesoru kvôli menšiemu množstvu otvorených spojení, zmenšuje záťaž na sieť vďaka tomu že nemusíme zasielať inicializačné dáta pre každý požiadavok[8].

Tento princíp nám umožňuje opakovať kroky 2 až 4, čo ale spôsobuje problémy pri meraní doby odozvy, z dôvodu že ak klient nebude pre každý požiadavok nové spojenie, meranie takéhoto trvalého spojenia bežným spôsobom bude ukazovať nesprávné hodnoty časov odozvy, keďže bude považovať viacero požiadavkov za jeden. Problém vznikne aj pri meraní ostatných metrík výkonnosti pretože v jednom spojenom zázname bude týchto hodnôt viac, pravdepodobne pre každý požiadavok v spojení jedna.



Obrázek 2.1: Graf HTTP spojenia

Piggybacking

Niekedy sa pri generovaní odpovede servera klientovi stane, že serveru trvá krátky čas po prijatí požiadavku vygenerovať odpoveď. V takomto prípade by bolo zasielanie potvrdenia o prijatí požiadavku a odpovede osobitne zbytočné, výhodnejšie je zaslať obe položky v rámci jedného paketu. Takýmto postupom šetríme procesor, ktorý generuje polovičný počet paketov, a takisto aj sieťovú záťaž z rovnakého dôvodu. Táto technika sa nazýva *piggybacking*, jej názov je odvodený od toho že potvrdzovací packet sa „zvezie“ spolu s dátovým paketom.

Táto metóda umožňuje spojiť kroky 3a a 3b do jedného. Monitorovanie výkonnosti nám ovplyvní tento princíp iba minimálne, avšak pri riešení výsledného produktu bolo potrebné brať tento princíp do úvahy pri rátaní doby odozvy.

Polia v hlavičke HTTP paketu

Hlavičky požiadavkov aj odpovedí HTTP paketov, zaslaných medzi klientom a serverov, obsahujú páry mien a hodnôt. Hodnoty v hlavičkách slúžia na predávanie správ medzi klientom a HTTP serverom, napríklad môžu obsahovať verziu HTTP ktorú klient požaduje alebo informácie o prehliadači ktorý užívateľ používa. Tieto informácie slúžia serveru aby mohol lepšie vygenerovať pre klienta odpoveď. Server zasiela klientovi takisto informácie v hlavičke, tie slúžia zvyčajne na popis toho čo čaká klienta v odpovedi, môže sa jednať napríklad o status kód podľa ktorého prehliadač zistí presmerovanie či o typ obsahu ktorý bol zaslaný v odpovedi[7].
V rámci tejto práce som musel naštudovať obsahy niektorých polí v hlavičke, jednalo sa o tieto:

- Content-Type (odpoveď) - MIME typ obsahu, zaslaného v odpovedi

- Status Code (odpoveď) - kód označujúci stav odpovede, kódy sú definované organizáciou IETF
- User-Agent (požiadavok) - informácie o klientovi (zvyčajne sa jedná o prehliadač)
- Host (požiadavok) - doménové meno serveru na ktorý sa chce klient pripojiť, pridáva sa z dôvodu že na jednej IP adrese môže byť viacero serverov (virtuálny hosting)

V rámci tejto práce som využil iba tieto polia, nemá preto zmysel definovať všetky.

Kapitola 3

Nástroje pre tvorbu web monitora

Táto kapitola bude bližšie pojednávať o princípoch a nástrojoch ktoré som využil pri tvorbe výsledného softvéru. Prvu časť tejto kapitoly tvoria základné informácie o NetFlowe ktorý bol využitý ako zdroj dát pre modul. Druhá časť kapitoly hovorí o frameworku Nemea na ktorom bol modul postavený.

3.1 NetFlow

Pre monitorovanie sietí bolo navrhnutých a vyvinutých už mnoho metód, každá z nich malá inú základnu myšlienku a iný účel. Jednou z týchto metód je **NetFlow**. NetFlow slúži ako nástroj pre pasívne monitorovanie siete, keďže sám nezasiela žiadne dáta do siete, namiesto toho iba zachytáva pakety vytvorené inými zariadeniami v sieti. Narozdiel od iných princípov pre pasívne monitorovanie však nespracúva všetky zachytené pakety, miesto toho ich agreguje (združuje) do tokov čím sa stáva tento prístup vhodnejší do vysokorýchlostných sietí, keďže znižuje záťaž na hardvér pre analýzu a ukladanie dát[9].

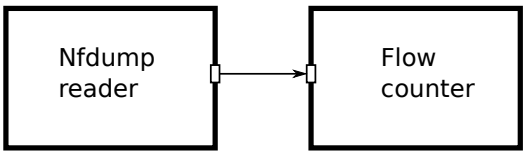
Architektúra NetFlow-u

Zakladnými prvkami systému NetFlow sú *exportér*, *kolektor* a *protokol*.

Exportérom rozumieme sieťové zariadenie alebo softvér ktorý monitoruje sieťovú prevádzku ktorá cez neho prechádza. Jeho ulohou je analyzovať pakety prechádzajúce cez neho a vytvárať z nich záznamy o sieťových tokoch.

Tok (flow) je definovaný ako sada IP paketov prechádzajúcich sledovaným bodom v určitom časovom okamihu s tým, že všetky pakety patriace určitému toku majú niekoľko spoločných vlastností. Tieto spoločné vlastnosti väčšinou zahŕňajú polia v hlavičke paketu ako sú zdrojová a cieľová IP adresa, porty a metainformácie. Potom čo exportér vytvorí záznam o sieťovom toku, je táto informácia odoslaná do NetFlow kolektora.

NetFlow kolektor je často realizovaný vo forme sotvéru bežiaceho na serveri, ktorý prijaté dáta buď ďalej spracúva a vytvára štatistiky o sieťovej prevádzke, alebo iba ukladá kvôli potrebe logovať správanie užívateľov na sieti. Dáta medzi exportérom a kolektorom sú prenášané pomocou komunikačného protokolu NetFlow vo forme záznamov. Od verzie 9 je možné okrem základných informácií o toku, ako je zdrojová a cieľová adresa toku, protokol pridať ďalšie informácie ktoré môže kolektor využiť na zlepšenie výpovednej hodnoty analýzy záznamov, avšak jednotlivé položky mohli byť iba v obmedzenej veľkosti[4]. Toto obmedzenie podnietilo vznik protokolu *IPFIX (Internet Protocol Flow Information Export)* ktorý umožnil prenos neobmedzene veľkých položiek[3].



Obrázek 3.1: Minimálný príklad systému Nemea, obrázok prebratý z [2]

3.2 Nemea

Keďže v dnešnej dobe dochádza k útokom na sieťove služby, aplikácie či užívateľov omnoho častejšie než tomu bolo v minulosti, zvýšil sa takisto aj dopyt po zariadeniach umožňujúcích detekciu takýchto útokov. Mnoho týchto útokov je možné objaviť iba v prípade neustáleho monitorovania a analyzovania sieťovej prevádzky. Zvyčajný postup v prípade takéhoto monitorovania spočíva v ukladaní monitorovaných dát na pamäťové média a následnej analýze za účelom odhalenia anomálií v nich.

Problém nastáva v prípade veľkých sietí kedy je potrebné uložiť obrovské množstvá dát pre ďalšie spracovanie. Ďalšou nepríjemnosťou pri takomto spracovaní dát je to, že jednotlivé dáta sú uložené po kusoch a po týchto kusoch sú aj analyzované, čo spôsobuje že jednotlivé analýzy nemajú informácie získané z minulých analýz, to môže spôsobiť že niektoré útoky nebude preto možné detekovať. Z týchto dôvodov bol organizáciou CESNET, skupinou Liberouter, vyvinutý framework pre analýzu sieťových dát **Nemea (Network Measurements Analysis)**.

Výhodou frameworku Nemea oproti bežným postupom je to, že spracúva dáta v reálnom čase, čím odbúrava potrebu ukladať ich na pevný disk. S tým samozrejme súvisí aj analýza sieťového toku ako celku, čo umožňuje odhaliť aj útoky prebiehajúce dlhšiu dobu.

Tvorba analyzátorov vo frameworku Nemea

Nemea je framework ktorý umožňuje vytváranie analyzátorov sieťových tokov v reálnom čase.[2] Systém analyzátorov je postavený z *modulov*, ktoré sú navzájom prepojené vstupnými/výstupnými rozhraniami *TRAP (Traffic Analysis Platform)*. Jednotlivé moduly sú navzájom nezávislé, pararelne bežiace jednotky, ktorých úlohov je väčšinou spracovanie a analýza dát ktoré do nich boli vložené buď cez rozhranie od iného modulu, alebo z nejakého iného zdroja. Spracované dáta sú z nich poslané do ďalšieho modulu, alebo sú výsledky analýzy zobrazené na výstupe. To umožňuje skladať rozsiahle systémy z menších logických blokov, prípadne jednoducho zreťaziť rôzne detektory útokov do jedného systému a takisto môžme jednotlivé bloky jednoducho nahradzovať napr. zmeniť vstup z živých dát na čítanie zo zdrojového súbora, pričom jednotlivé modely môžú byť pridávané a odoberané zo systému dynamicky počas behu systému.

TRAP rozhrania prepájajúce moduly umožňujú jednosmerný prenos informácií vo forme záznamov v špecifickom formáte *UniRec (Unified Record)*. Každý UniRec záznam sa skladá z niekoľkých polí ktoré sú definované menom a typom. Sada týchto polí sa nazýva predloha a každé TRAP rozhranie môže používať iba jednu predlohu na zasielanie alebo príjmanie dát, cez jedno rozhranie môže prejsť iba jeden druh správ. Najbližšie by sa dal prirovnať UniRec záznam k štruktúre v jazyku C. Minimálný príklad takého systému je zobrazený na obr. 3.1, kde môžme vidieť systém zložený z dvoch modulov, pričom z jedného modulu sú dáta zasielané do druhého cez TRAP rozhrania.

3.3 Framework Django

Django je bezplatný, open-source *MVC (Model-View-Controller)* framework na tvorbu web aplikácií. Je písaný v programovacom jazyku Python a jeho hlavnou úlohou je čo najviac zjednodušiť tvorbu webu spojenú s komunikácou s databázou, pričom sa snaží zachovať dobré programatorské praktiky.

3.3.1 Django MVC

Každá stránka postavená na frameworku Django sa skladá z troch základných komponentov. Pri iných frameworkoch sa zaužívalo ich označenie ako Model, View (zobrazovač), Controller (ovládač).

Úlohou modulu je komunikácia s databázou, väčšinou za úkolom výberu dát, ktoré sú následne zaslané zobrazovaču ktorý ich upraví do podoby ktorú zobrazí užívateľovi. Ovládač následne reaguje na akcie vykonané užívateľom a pomocou nich ovplyvňuje správanie zobrazovača a modelu.

Django sa nedrží tohto vzoru úplne presne, ale dosť nato aby sa dal považovať za MVC framework. V prípade Djanga sa však využíva vzor so skratkou MTV (Model-Template-View). Dôvod pre tento vzor je to, že väčšina akcií ktorú spracúval ovládač je v Djangovi implementovaná automaticky. Rozdiel medzi MVC a MTV je v tom že v prípade Djanga slúži vrstva zobrazovača len ako prepoj medzi modelom a šablónu. Šablóna potom umožňuje vybrať ako budú jednotlivé dáta zobrazené[10].

Kapitola 4

Návrh a implementácia

Táto kapitola bude popisovať návrh a realizáciu produktu ktorý bol definovaný zadaním. Začnem túto kapitolu mojím pochopením toho, ako by mal vyzerať výsledný produkt podľa zadania, ďalej je potrebné zmieniť skutočnosti vyplývajúce zo zadania. Následovať bude môj návrh jednotlivých častí systému a kapitolu zakončím popisom konečnej implementácie jednotlivých častí.

4.1 Zadanie

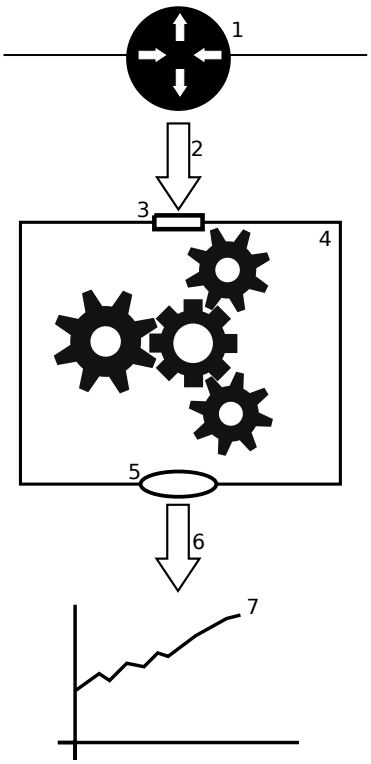
Z tretieho bodu zadania nám vyplýva že výsledkom našej práce by mal byť naprogramovaný funkčný modul do systému Nemea. Tento modul by mal byť schopný získavať informácie od exportérov NetFlow dát na sieti. Ďalším krokom je získanie užitočných dát pre určenie výkonnosti HTTP serverov z prijatých záznamov. Získane dáta je potrebné zobraziť, čím bolo potrebné vytvoriť užívateľské rozhranie z ktorého by bolo možné vyčítať údaje o výkonnosti monitorovaných zariadení.

4.2 Skutočnosti vyplývajúce zo zadania

Cieľom tejto práce je vytvoriť modul do systému Nemea. Z toho plynie niekoľko skutočností ktorých sme sa museli držať pri návrhu. Moduly systému Nemea sú typicky písané v programovacom jazyku C prípadne C++. Z dôvodu že s jazykom C mám väčšie skúsenosti som zvolil tento jazyk ako implementačný pre modul. Nemea moduly komunikujú medzi sebou cez TRAP rozhrania, preto bolo potrebné pre všetky NetFlow dáta s ktorými som chcel pracovať v module vytvoriť vstupné rozhranie. Kvôli živým vstupným dátam bolo potrebné testovať modul na serveri benefizio na ktorom beží operačný systém Linux, časom sa presunulo aj samotný vývoj programu priamo na tento server.

4.3 Návrh

Táto sekcia popisuje návrh jednotlivých častí výslednej aplikácie. Nesnažím sa podrobne popisovať každý aspekt výsledného produktu, skôr popisujem iba základné myšlienky a problémy ktoré som musel riešiť pri návrhu a ktoré ovplyvnili konečnú formu produduktu. Ako prvé popíšem celkový návrh architektúry modulu z hľadiska modulu ako časti systému Nemea, pokračovať budem blokom o spracovaní dát a mojimi pokusmi o zmenšenie pamäťovej záťaže modulu. Nakoniec popíšem možnosti pri zobrazovaní výsledkov, keďže bez tejto časti



Obrázek 4.1: Grafické znázornenie návrhu aplikácie, 1. Exportér, 2. Záznamy o toku, 3. Vstupné rozhranie, 4. Modul, 5. Výstupné rozhranie, 6. Výstupné dáta, 7. Grafická reprezentácia výsledkov

by užitočnosť modulu utrpela. Pre ilustráciu som vytvoril grafickú formu výsledného návrhu viď obrázok 4.1. Je v nej možné vidieť exportér umiestnený na smerovači v sieti ktorý analyzuje pakety prechádzajúce ním, vytvára z ních záznamy a zasiela ich do môjho modulu. Modul záznamy analyzuje a napokon sú výsledky monitorovania zobrazené vo vhodnej forme.

4.3.1 Architektúra modulu

Prvá časť ktorú bolo treba navrhnúť bola architektúra modulu, čo znamená počet vstupných a výstupných rozhraní, a UniRec šablóny na jednotlivých rozhraniach. Možností som nemal veľa, keďže zo zadania vyplynula potreba minimálne jedného vstupného rozhrania pre príjmanie dát od exportéra. Čo sa týka výstupných rozhraní, tam som takisto mohol zvoliť buď úplne vynechať výstupné rozhranie alebo pridať jedno rozhranie pre zapojenie ďalšieho modulu.

Pri výbere šablóny som mohol spočiatku rozhodovať iba medzi šablónami poskytovanými frameworkom Nemea a to `BASIC_FLOW` prípadne `COLLECTOR_FLOW`. Obe tieto šablóny poskytujú iba základné informácie o toku, čo ma viedlo k využitiu pluginu pre exportér, ktorý okrem iného zvačšil množstvo informácií, ktoré poskytovala šablóna o informácie špecifické pre HTTP, jedná sa o položky s predponou `HTTP_SDM`.

4.3.2 Proces spracúvania záznamov

Ďalším problémom bolo samotné spracovávanie dát. Spracúvanie dát bolo možné rozdeliť do troch krokov.

1. Filtrácia
2. Analýza
3. Ukladanie

Filtrácia

Keďže je pravdepodobné že nie všetky dáta ktoré zachytí exportér a odošle na vstup modulu sú tie o ktoré má užívateľ záujem či už z dôvodu že, ako v mojom prípade exportér zachytáva veľké množstvo dát z rôzných miest (exportéry zachytávali dáta zo sônd rozmiestnených v rámci siete CESNET), alebo má užívateľ záujem iba o vybrané zariadenia v sieti, bolo nevyhtnutné navrhnúť filtrovanie záznamov ktoré modul spracuje. V tomto prípade bolo potrebné vymyslieť údaje podľa ktorých bude filter rozhodovať či daný záznam bude ďalej analyzovať alebo ho zahodí a zdroj filtrovacích pravidiel. Čo sa týka rozhodovania o filtrácií možnosťou bolo vytvoriť buď pridávajúce alebo vylučujúce pravidlá (include/exclude) pričom tieto pravidlá sa mohli aplikovať na základe IP údajov záznamu. Medzi tieto údaje patrí IP adresa servera, klienta, porty použité pri komunikácií, protokol atď.

Analýza

Potom čo bolo filtrovaním zaistené že sa k analýze dostanú iba dáta o ktoré máme záujem, nasledovala samotná analýza dát. Tá spočívala v tom že modul, z dát ktoré získal od exportéra, vybral údaje ktoré sú potrebné pre monitorovanie výkonu HTTP serveru a vhodne ich uložil. Metriky pre monitorovanie HTTP serverov sú zvyčajne založené na meraní času medzi dvoma akciami napr. prijatie požiadavku a odoslanie odpovede, takisto je ale možné získavať vhodné údaje z monitorovanie počtu unikátných IP adries, prípadne ak máme prístup k hlavičke HTTP paketu tak je možné ukladať zaujímavé údaje z nej napr. počet zaslaných chybových hlášok alebo typ prenášaného obsahu. Získané údaje záviseli od exportéra a šablóny použitej na vstupné rozhranie, pri rozhodovaní som musel brať do úvahy pamäťovú náročnosť aplikácie, keďže každý údaj zvyšuje pamäťové nároky pre každý záznam, ktorých môže byť počas behu aplikácie niekoľko stoviek až tisíciek.

Ukladanie

Posledným krokom spracovania bolo ukladanie analyzátorom vytvorených záznamov do dátovej štruktúry. V tejto časti bolo potrebné vyriešiť problém s trvalými záznamami, viac informácií o trvalých záznamoch viď 2.3. Až doteraz bolo modulu jedno v akom poradí budú dáta prichádzať, ale keďže pre vyrátanie správných metrík bolo potrebné spájať záznamy patriace do jedného spojenia. Kvôli tomuto problému bolo potrebné navrhnúť riešenie ktoré by umožňovalo ukladať záznamy o spojeniach a do týchto záznamov pridať informácie o jednotlivých požiadavkoch klienta a prislúchajúcích odpovediach servera. Nieje zaručené, respektíve je vysoko nepravdepodobné, že exportér nám odošle takúto komunikáciu ako niekoľko, po sebe idúcich záznamov. Takmer určite sa pri príjmaní záznamov v rámci jedného spojenia pararelne budú zasielať záznamy z iných spojení. Týmto nám vznikla pri ukladaní nutnosť vyhľadávať v uložených záznamoch zhodu, aby sme mohli do spojenia, od ktorého

už máme niekoľko predchádzajúcich záznamov, pridať aj aktuálny. Výsledná dátova štruktúra musela byť teda prispôsobená týmto požiadavkám ak mala byť schopná správne ukladať záznamy.

4.3.3 Znižovanie pamäťových nárokov

V tejto fáze návrhu, bol navrhnutý produkt schopný príjmať dáta od exportéra, filtrovať a analyzovať ich. Výsledky analýzy bol následne schopný ukladať do dátovej štruktúry. Avšak nato aby bol produktu použiteľný v skutočnom prostredí bolo potrebné pridať ešte niekoľko častí.

Pri aplikácií ako je táto, ukladajúca analýzu nekonečného toku dát, je potrebné rátať s tým že je na jej beh potrebné vyhradiť určité množstvo pamäte s ktorou bude pracovať. Aby som čo najviac znížil pamäťové nároky svojho modulu, bolo potrebné navrhnúť riešenia ktoré by zmenšili množstvá údajov ktoré musí mať aplikácia v pamäti v určitý čas. Na druhej strane je ale potrebné, aby táto redukcia dát neovplyvnila výslednú analýzu vo veľkej miere. V tomto smere som napokon navrhol dve opatrienia a to selekcia záznamov a agregáciu záznamov.

Selekcia záznamov spočíval v tom že bolo potrebné vybrať záznamy ktorých informačná hodnota výkonu serveru bola nižšia a tým pádom boli odstránené aby nezaberali miesto iným záznamom z ktorých sa dá lepšie vyčítať výkon serveru. Z toho dôvodu bolo potrebné odstraňovať záznamy ktoré boli zaslané bez inicializácie spojenia, príkladom takýchto záznamov sú záznamy z komunikácie ktorá začala pred spustením modulu, keďže ich výpovedná hodota môže byť znížená kvôli tomu, že nieje možné si pozrieť celú komunikáciu. Ďalej bolo potrebne mazať záznamy ktoré nesprávne zobrazovali odozvy servera, pravdepodobne z dôvodu na klientskej strane. Za takéto záznamy som považoval všetky záznamy ktorým vypršal časový limit komunikácie predtým, ako bola uzavretá, čo môže znamenať že klient násilne ukončil komunikáciu a znova, tieto hodnoty by sa nemali brať do úvahy.

Agregácia záznamov spočívala v tom zmenšiť množstvo uložených záznamov pričom sa musí zachovať ich výpovednú hodnotu. Možností ako toto dosiahnúť je viacero, môžme vytvoriť priemer hodnôt za určité časové obdobie a nahradiť tak viacero záznamov jedným, prípadne vytvoriť novú dátovú štruktúru do ktorej vložíme výsledné dáta ktoré by inak bolo potrebné rátať zo záznamov.

Zobrazovanie výsledkov analýz

Dáta uložené v štruktúre jazyka C niesú najvhodnjšie na reprezentáciu nameraných údajov, preto bolo potrebné vymyslieť spôsob, akým by bolo možné tieto dáta reprezentovať v ľudsky prijateľnej forme. Možností sa v tomto prípade ponúka mnoho, napríklad je možné uložené dáta vyexportovať v tabuľkovom formáte a nechať užívateľa nech si zvolí či naprogramuje vlastný nástroj pre zobrazenie týchto dát alebo využije existujúce riešenie.

Ďalšou možnosťou je zobrazovať výsledky meraní cez grafické rozhranie vytvorené pomocou knižnice jazyka C na tento účel vhodne zvolenou. Výhodou týchto dvoch riešení je to že v prvej možnosti nieje potrebné programovať žiadné rozhranie, keďže použité rozhranie by bol štandardný výstup programu. Pri použití druhej možnosti je potrebné naprogramovať iba rozhranie iba medzi dvoma kódmi v jazyku C, čo je väčšinou veľmi jednoduché. Nevýhodou by bolo to že by bolo potrebné programovať funkcie pre zoraďovanie, agregácie a výber dát v jazyku C, ktorý na to nieje práve vhodne uspôsobený. Aj ked nie úplne problém, ale skôr mne nesympatické, je pri týchto riešeniach to, že prípadný užívateľ by musel vedieť

pracovať v Linuxe, čo by možno odradilo potenciálnych používateľov od zvolenia si tohto modulu.

Treťou možnosťou je kombinácia predošlých dvoch, dáta budú exportované avšak zároveň budú zobrazované cez mnou vytvorenú aplikáciu. Toto riešenie umožňuje užívateľovi, v prípade potreby, vytvoriť si vlastný nástroj na zobrazovanie výsledkov sledovania. Avšak v prípade že bude chcieť jednoduché riešenie môže využiť moju aplikáciu. Nevýhodou je potreba naprogramovať rozhranie medzi modulom v C a iným prostredím, a takisto potreba ďalšej aplikácie slúžiacej ako užívateľské prostredie.

4.4 Implementácia

Táto kapitola popíše realizáciu návrhu z predošlej kapitoly. Postup krokov bude približne rovnaký aký bol pri návrhu, najprv implementujem vstup pre svoj modul a popíšem využitie pluginu pre exportér, pokračovať bude popis pridania modifikovateľnosti modulu a filtrácie. Následne popíšem realizáciu jadra modulu, procesu spracúvajúceho samotné dáta. Ďalšia časť sa bude venovať periodickým procesom a implementácií databázy a napokon popíšem výsledné riešenie zobrazovania vo forme webu.

4.4.1 Vstupné rozhranie

Rovnako ako pri návrhu, aj pri implementácii Nemea modulu je prvým krokom vytvorenie TRAP rozhraní pre komunikáciu modulu s okolitými modulmi. Napokon som sa rozhodol, kvôli jednoduchosti, implementovať iba jedno vstupné rozhranie. Toto rozhranie príjma dáta od exportérov, umiestnených na sondách ktoré sú rozmiestnené v sieti CESNET, toto riešenie bolo výhodné lebo som mohol počas implementácie testovať modul na reálných dátach, na druhej strane ale bolo nutné rátať s veľkým množstvom vstupných dát. Ako šablónu pre toto rozhranie som ako prvé zvolil základnú šablónu `BASIC_FLOW`. Táto šablona poskytuje základné informácie o toku, konkrétne sa jedná o tieto údaje:

`SRC_IP` - zdrojová IP adresa toku

`DST_IP` - cieľová IP adresa toku

`SRC_PORT` - zdrojový port toku

`DST_PORT` - cieľový port toku

`PROTOCOL` - komunikačný protokol toku

`TCP_FLAGS` - TCP príznaky toku

`PACKETS` - množstvo paketov z ktorých sa skladá tok

`BYTES` - veľkost dát z ktorých je tok v bytoch

`TIME_FIRST` - čas v príchodu prvého paket toku

`TIME_LAST` - čas v ktorom prišiel posledný paket toku

Avšak časom sa ukázalo že údaje poskytované touto šablónou boli nedostačujúce pre dobrú analýzu.

Plugin pre exportér

Niekedy v tom čase sa vynorila možnosť vylepšiť dáta prijaté z exportéra o údaje špecifické pre HTTP. Touto možnosťou bol plugin pre exportér vyvíjaný Jakubom Budiským. Dôvod pre použitie tohto pluginu bolo samozrejme to, že nám viac informácií umožňuje robiť lepšie monitorovanie HTTP serverov, ale aj preto, lebo riešil problémy ktoré spôsobovali servery využívajúce trvalé spojenia na komunikáciu. Problém s trvalým spojením spočíva v tom, že prípade že klient nevytvorí pre každý požiadavok nové spojenie, meranie bude ukazovať nezmyselné hodnoty, bližšie informácie o trvalých spojeniach viď 2.3. Po implementácií mnou navrhnutej zmeny týkajúcej sa merania času odozvy, bol plugin pripravený pre použitie v mojom riešení.

Funkcia pluginu spočívala v tom, že v prípade HTTP toku neukladal záznam o toku ako celok od otvorenia spojenia až po jeho uzavretie. Miesto toho vytvoril v prípade trvalého spojenia viacero záznamov o tomto toku. Delenie spojenia sa líšilo v závislosti od toho, či sa jednalo o správu od klienta serveru alebo naopak. V prípade že sa jednalo o komunikáciu od klienta k serveru rozdelil tok pred prvým zaslaným paketom požiadavku (Request) . Pri správach od servera klientovi rozdeľovanie nastalo pri prvom pakete odoslaného potvrdenia (ACK). Grafické znázornenie tohto procesu viď obrázok 4.2.

Druhá funkcia pluginu je, že z hlavičiek HTTP paketov, ktoré spracuje, vyberie niektoré údaje a spolu so základnými NetFlow údajmi ich pošle kolektoru. Toto umožnilo obohatiť šablónu `BASIC_FLOW` o tieto položky:

`HTTP_SDM_REQUEST_METHOD_ID` - ID metódy požiadavku (POST, GET...)

`HTTP_SDM_REQUEST_HOST` - doménové meno servera

`HTTP_SDM_REQUEST_AGENT` - agent klienta

`HTTP_SDM_RESPONSE_STATUS_CODE` - stavový kód odpovede servera

`HTTP_SDM_RESPONSE_CONTENT_TYPE` - typ obsahu v odpovedi servera

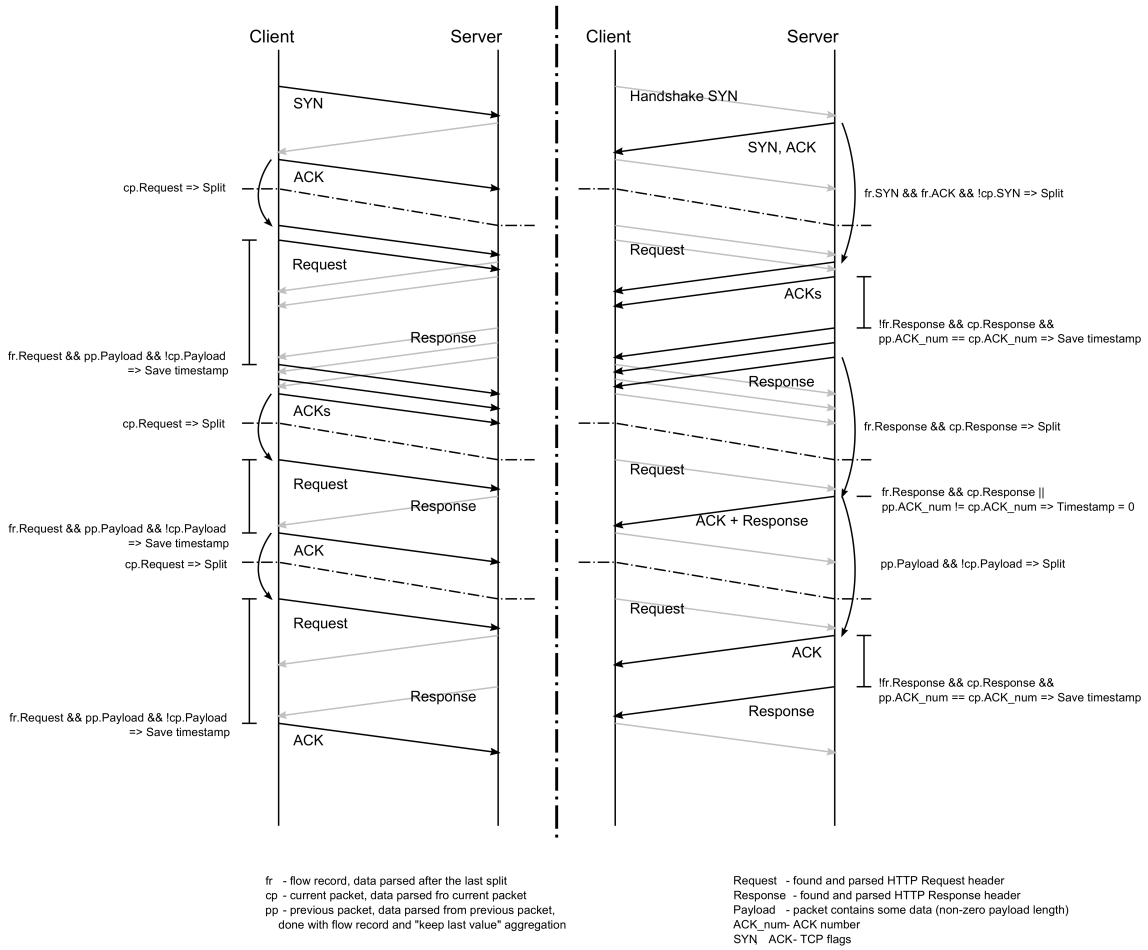
`HTTP_SDM_RESPONSE_TIME` - čas odozvy

Plugin umožňuje ešte zasielanie informácií o URL stránky ktorú klient požaduje a stránku z ktorej bol požiadavok vytvorený (referer), avšak kvôli tomu že tieto údaje môžu obsahovať citlivé dáta a na rozdiel od IP adries ich nieje možné jednoducho anonymyzovať, som tieto údaje nedostával. Čas odozvy reprezentoval rozdielné veci v závislosti od toho či sme ho rátali pri komunikácii od servera ku klientovi, alebo opačne. V prípade záznamov o komunikácii od klienta serveru reprezentuje čas medzi odoslaním posledného paketu požiadavku až po čas prvého paketu potvrdenia o prijati odpovede, kdežto pri komunikacii opačného smeru reprezentuje čas medzi prijatím posledného požiadavku a odoslaním odpovede, viď obrázok 4.2.

Tento proces spôsobí to, že v prípade že cez NetFlow exportér prejde jedno trvalé spojenie, od exportéru sa kolektoru pošle viacerov záznamov ktoré budú mať rovnaký zdroj a cieľ v krátkom čase, avšak nieje zaručené že pri príjmaní týchto záznamov sa medzi nimi nevyskytne záznam z inej komunikácie.

4.4.2 Konfigurácia

Keďže modul vyžadoval určitú úroven konfigurovateľnosti užívateľom, bolo nevyhnutné implementovať rozhranie cez ktoré by bolo možne predať parametre modulu, ktorý by podľa



Obrázek 4.2: Princíp delenia HTTP komunikácie pluginom, originál vytvorený Jakubom Budiským

nich následne upravil svoje správanie. Ako zdroj týchto parametrov som sa rozhdol použiť textový súbor preto, lebo použitie argumentov pri volaní modulu na predanie parametrov by síce bolo podstatne jednoduchšie na implementáciu, ale tieto parametre môžu byť rozsiahle a volanie programu s mnohými argumentami podľa môjho názoru nevyzerá esteticky pekne. Zároveň som nechcel užívateľa nútiť písať tieto parametre pri každom spustení modulu.

Kvôli tomuto rozhodnutiu som však bol nútený implementovať rozhranie ktoré by bolo schopné načítať konfiguráciu uloženú v textovom súbore, kde jednotlivé hodnoty nastavení budú v určitom formáte, tieto hodnoty rozložiť a previesť ich na premenné použiteľné v zdrojovm kóde.

K načítaniu konfiguračného súboru som sa rozhodol nepísať sí vlastné riešenie, keďže mi to prišlo zbytočné a nepraktické, miesto toho som sa rozhodol pre použitie knižnice `libconfig`.

Táto knižnica umožňuje načítavanie a rozklad štruktúrovaných súborov s nastaveniami, jediná podmienka ja tá, že hodnoty v tomto súbore musia byť zapísané vo formáte definovanom knižnicou. To však nebol problém, keďže som nemal žiadne preferencie formátu konfiguračného súboru a knižnica umožňovala prehľadný zápis všetkých nastavení ktoré som požadoval.

Použitie knižnice je jednoduché, stačí predať do jednej z jej funkcií názov súboru, funkcia načíta hodnoty z daného súboru, prípadne upozorní na chybu v prípade nesprávnej syntaxe v súbore. Hodnoty zo súboru sú načítané do špeciálnej dátovej štruktúry `config_setting` nad ktorou už potom stačí iba volať ďalšie knižničné funkcie ktoré vrátia hodnotu nastavenia definovanej názvom ktoré sme predali tejto funkcií. Knižnica umožňuje združovať nastavenia do väčších pomenovaných blokov, čo umožňuje vytvoriť rovnaké konfigurácie pre množstvo prvkov, jednoducho si to možno predstaviť ako objekty v OOP, kde viacero inštancií jedného objektu môže mať rôzne parametre. Táto funkcionalita mi umožnila vytvárať konfigurácie pre filtre, čo v konečnom dôsledku umožňuje užívateľovi definovať ľubovoľne množstvo filtrov.

Pre to, aby sme vedeli ako načítavať hodnoty z konfiguračného súboru, bolo potrebné definovať šablónu tohto súboru. Predpokladom pre túto šablónu bolo že pomocou nej mal byť užívateľ schopný definovať dve veci.

Prvá z nich sú všeobecné parametre programu, napr. príznak označujúci úlohy ktoré sa majú vykonať pri spustení programu, periódy opakujúcich sa procesov. Vzor pre tieto nastavenia je `Parameter = Hodnota;` kde *Parameter* je možné vybrať z predom definovaných hodnôt, reprezentujúcich rôzne nastavenia, *Hodnota* závisí od parametra ktorý nastavujeme, podľa kontextu to môže byť tradična Booleovská hodnota (true/false), prípadne číslo či slovo.

Týmto spôsobom bolo možné vytvoriť nastavania pre periódy exportácie dát, ktorým je možné prispôsobiť rozdelenie záťaže modulu na systém medzi databázu a pamäť servera. Takisto je možné nastaviť periódy mazania a agregácie čo spôsobí odstránenie záznamov, ale na druhej strane znižuje pamäťové nároky. Dôležitým nastavením je nastavenie núdzového režimu, ktoré sa spustí v prípade že prekročíme určitý počet záznamov a ktorého úlohou je vykonať premazávanie záznamov s trochu prísnejšími kritériami než má periodické premazávanie.

Druhé nastavenie, ktoré bolo potrebné definovať, sú filtre rozhodujúce o spracovaní či nespracovaní prichádzajúceho záznamu. Keďže som chcel dať užívateľovi možnosť definovať viac ako jeden filter, bolo potrebné to vziať to do úvahy aj pri tvorbe šablóny, čím bežný prístup s jednou hodnotou pre jeden parameter samozrejme nestačil. Vďaka tomu že knižnica podporuje uzatváranie konfigurácií do blokov, bola napokon vytvorená šablóna, kde každé

pravidlo je definované názvom a údajmi, ktorých zhoda s záznamom rozhodovala o tom či bude dané pravidlo aplikované na záznam. Ako údaje pre tento proces bola napokon vybraná kombinácia IP adresy servera a masky pomocou ktorej je možné definovať IP rozsah, čo umožňovalo definovať viac ako jedno zariadenie pre pravidlo. Ďalšia položka ktorá je v týchto pravidlách je špecifikácia postupu pri spracovávaní záznamu, v mojom prípade som tam pridal možnosť zrušiť ukladanie informácií o type obsahu (Content-type) záznamu. Vzor pre nastavenia filtrov vyzerá takto:

```
server =
{
        nazov =
        {
                IP = "IP_servera";
                maska = maska_rozsahu;
                watch = true/false;
                content = true/false;
        };
};
```

0..X

IP servera je IPv4 alebo IPv6 adresa vo formáte string, *maska* reprezentuje rozsah siete na ktorý chcem aplikovať pravidlo a prepínače *watch* a *content* hovoria či má modul záznam ktorý je v rozsahu monitorovať, to preto aby bolo možné vytvárať pridávacie aj vylučovacie pravidlá, a či máme od daného záznamu ukladať informácie o type obsahu.

Posledným problémom ktorý bolo potrebné v súvislosti s konfiguráciou vyriešiť bolo ukladanie načítanej konfigurácie do vhodnej štruktúry, aby sme mohli využiť informácie z konfiguračného súboru v kóde. Kvôli tomu že užívateľom definované pravidlá pre filtrovanie boli použité iba na jednom mieste, kdežto ostatné nastavenia mohli byť použité na rôzných miestách v programe, rozhodol som sa vytvoriť rozdielné dátové štruktúry pre tieto dve nastavenia.

Dátovú štruktúru pre všeobecné nastavenia som chcel vytvoriť tak, aby bolo možné pridávať dodatočné nastavenia čo najjednoduchším spôsobom. Preto som zvolil napokon štruktúru jazyka C, z ktorej je vytvorená jedna inštancia a ktorá obsahuje položku pre každé nastavenie. Táto štruktúra je potom predávaná do funkcií ktorých správanie môže ovplyvniť.
Pre nastavenia filtra bolo potrebné vziať do úvahy že týchto nastavení môže byť viacero ale na druhej strane budú použité iba na jednom mieste a tým je filter. Napokon som pre túto funkcionalitu zvolil ukladanie každého pravidla pre filter do štruktúry `user_configuration` ktorá obsahuje informácie o IP adrese, maske a príznaky pre modifikáciu spracovania záznamov ktoré majú pasujú na údaje v danom pravidle. Všetky tieto pravidlá su následne za sebou vo forme zoznamu.

4.4.3 Filtrácia

Filtrovanie je v mojom module implementované vo dvoch krokoch. Ako prvé je potrebné zaručiť že záznam ktorý príde od exportéra je skutočne záznam o HTTP komunikácií, keďže o ostatné záznamy nemá modul záujem. Za druhé je potrebné overiť že sa jedná o záznamy komunikácie na zariadeniach ktoré užívateľ chcel sledovať, tj. zariadenia definovnaé v konfiguračnom súbore.

Pravidlá prvého filtrovanie sú veľmi priamočiare, keďže poznáme parametre HTTP komunikácie a stačí nám vylúčiť tie záznamy, ktoré tieto podmienky nespĺňajú, jedná sa teda iba o kontrolu jedného z portov, ktorý musí byť HTTP port servera a protokolu, ktorý musí

byť TCP, viď 2.3. Táto kontrola prebieha hneď po príjatí záznamu, pretože nemá zmysel ďalej pracovať s inými ako HTTP záznamami.

Druhý krok filtrovania kontroluje či prijatý záznam patrí do komunikácie medzi zariadeniami definovanými užívateľom. Táto kontrola sa vykonáva tým, že sa z prijatého záznamu vyberú údaje pomocou ktorých je vykonávaná filtrácia, čiže IP adresa zariadenia ktoré bolo identifikované ako server, a porovnáva sa s uloženými filtrami. Porovnávanie vracia najkonkrétnejšiu zhodu, konkrétnosť zhody závisí od masky pravidla v ktorom bola nájdená zhoda. To umožňuje vytvoriť obecné pravidlá pre zariadenia v podsieti, ale pre určité zariadenia ich zmeniť podľa potreby.

Ukázalo sa že porovnávanie dvoch IP adries, pričom je potreba brať ohľad na maskovanie, je problematické, pretože framework Nemea nepodporuje takéto porovnávanie. Z toho dôvodu bolo potrebné naprogramovať funkciu ktorá by s použitím ostatných funkcií frameworku takéto porovnávanie umožňovala. Riešenie napokon využilo bitové posuny a XOR porovnania binárnej formy dvoch adries, pričom bolo potrebné brať do úvahy či sa jedná o IPv4 alebo IPv6 adresy.

4.4.4 Analýza a ukladanie

Analýza v tomto bode spočívala iba v tom, že model vylúčil záznamy ktoré prichádzali bez kontextu, tj. záznamy ktoré neoznačovali prvý paket toku pričom pred nimi neboli prijaté žiadné iné pakety z toku, a takisto záznamy ktoré prišli po prijatí posledného paketu toku. Toto bolo iba opatrenie pre ušetrenie miesta a pre to aby bolo zabránené takýmto „nesprávnym“ záznamom ovplyvniť výsledok monitorovania. Pozíciu paketu v toku, bolo možné zistiť pomocu TCP príznakov prijatého toku. Exportér zasiela príznaky toku ako zlúčenie príznakov všetkých paketov v danom toku, takže stačilo vyhľadávať záznamy s príznakom SYN pre počiatok a FIN pre koniec spojenia.

Potom čo sme mali filtrovaním a analýzou zabezpečené správne dáta pre monitorovanie, bolo potrebné rozhodnúť ako bude monitorovanie vyzerať. Vo všeobecnosti monitorovanie spočíva v ukladaní takých dát počas prevádzky, aby z nich mohol užívateľ získať vedomosti ktoré potrebuje. Ako som už riešil pri návrhu, ukladať si všetky dáta je nepraktické, napokon som teda zvolil ukladať iba zdrojovú a cieľovu IP adresu a port, všetky časy ktoré môžme získať z exportéru (začiatok, koniec komunikácie a odozvu), informáciu o obsahu ak to užívateľ nešpecifikoval ináč a status kód.

Ukladanie týchto dát som napokon robil do dvoch rôzných dátových štruktúr. Dôvod prečo som to implementoval týmto spôsobom bol ten, že pre jednu komunikáciu môže byť vytvorených viacero záznamov ale každý záznam patrí iba do jednej komunikácie. Vytvoril som teda jednu dátovú štruktúru pre ukladanie spojení a do nej som vložil druhú dátovú štruktúru ktorá ukládala jednotlivé časti tejto komunikácie. Pre každú novú komunikáciu bola teda vytvorená nová inštancia štruktúry, a pre každý ďalší záznam bolo potrebné vyhľadať existujúcu inštanciu a pridať k nej prijatý záznam.

Riešenie ktoré som napokon implementoval pre záznamy o komunikácií bola hashovacia tabuľka, kvôli tomu že umožňuje rýchle vyhľadávanie pokiaľ poznáme kľúč podľa ktorého chceme hľadať. Kľúčom v tomto prípade bol identifikátor spojenia, čiže IP adresy a porty na ktorých existuje dané spojenie. Keďže jazyk C neposkytuje hashovacie tabuľky, bolo potrebné buď si túto funkcionalitu naprogramovať, alebo využiť nejaké existujúce riešenie. Pretože mi prišlo zbytočné vytvárať nové riešenia tohto obecného problému, rozhodol som sa pre tento účel využiť knižnicu `uthash`. Hlavný dôvod prečo som zvolil práve túto knižnicu je ten, že umožňuje vytvárať hashovacie kľúče zo štruktúr jazyka C. Vďaka tejto možnosti

netrebalo spájať identifkátory komunikácie do rozsiahlých neprehľadných kľúčov, miesto toho som pre každý prichádzajúci záznam vytvoril štruktúru `record_key` v ktorej sú tieto položky uložené. Túto štruktúru som potom predal funkcii knižnice, ktorá mi, buď vrátila už existujúcu položku, čo znamenalo že prijatý záznam nieje prvý a stačí ho pridať do existujúcej komunikácie, alebo nevrátila nič, vtedy som štruktúru využili ako kľúč pre nový záznam o komunikácií. Jednotlivé časti komunikácie som realizoval ako lineárny zoznam, kde bol každý prijatý záznam reprezentovaný ako jedna položka zoznamu.

Ukladanie typu obsahu

Poslednú vec ktorú bolo treba vyriešiť bolo ukladanie typu obsahu prijatého záznamu. Typ obsahu získavá modul ako polžku od exportéru v textovej forme a keďže som nechcel pre každú prijatú hodnotu alokovať novú pamäť, rozhodol som sa ukladať iba prvý unikátny typ, do vhodne zvolenej štruktúry, a pri ostatných vrátit iba číselnú reprezentáciu (ID), ktorá sa vloží do štruktúry reprezentujúcej jednotlivé položky komunikácie.

Na tento účel som potreboval dátovú štruktúru v ktorej by bolo možné vyhľadávanie podľa textu, a keďže som už mal podobné riešenie implementované v súvislosti s ukladaním záznamov o komunikácii, rozhodol som sa znova využiť hashovaciu tabuľku v knižnici `uthash`. Tentokrát som ako kľúč využil textovú reprezentáciu názvu typu obsahu. V prípade zhody funkcia vrátila existujúcu číselnú reprezentáciu daného typu obsahu v tabuľke, v prípade že neexistovala zhoda, bolo potrebné vytvoriť novú položku v tabuľke a vygenerovať ID.

Výsledok tohto procesu sú uložené záznamy o tokoch prechádzajúcích exportérom, pričom je možnosť vybrať si ktoré zariadenia chceme sledovať. Ilustrácia aktuálneho procesu od získania záznamov až po ukladanie viď obrázok 4.3. Ďalším krokom bolo vytvoriť určitý druh reprezentácie uložených hodnôt.

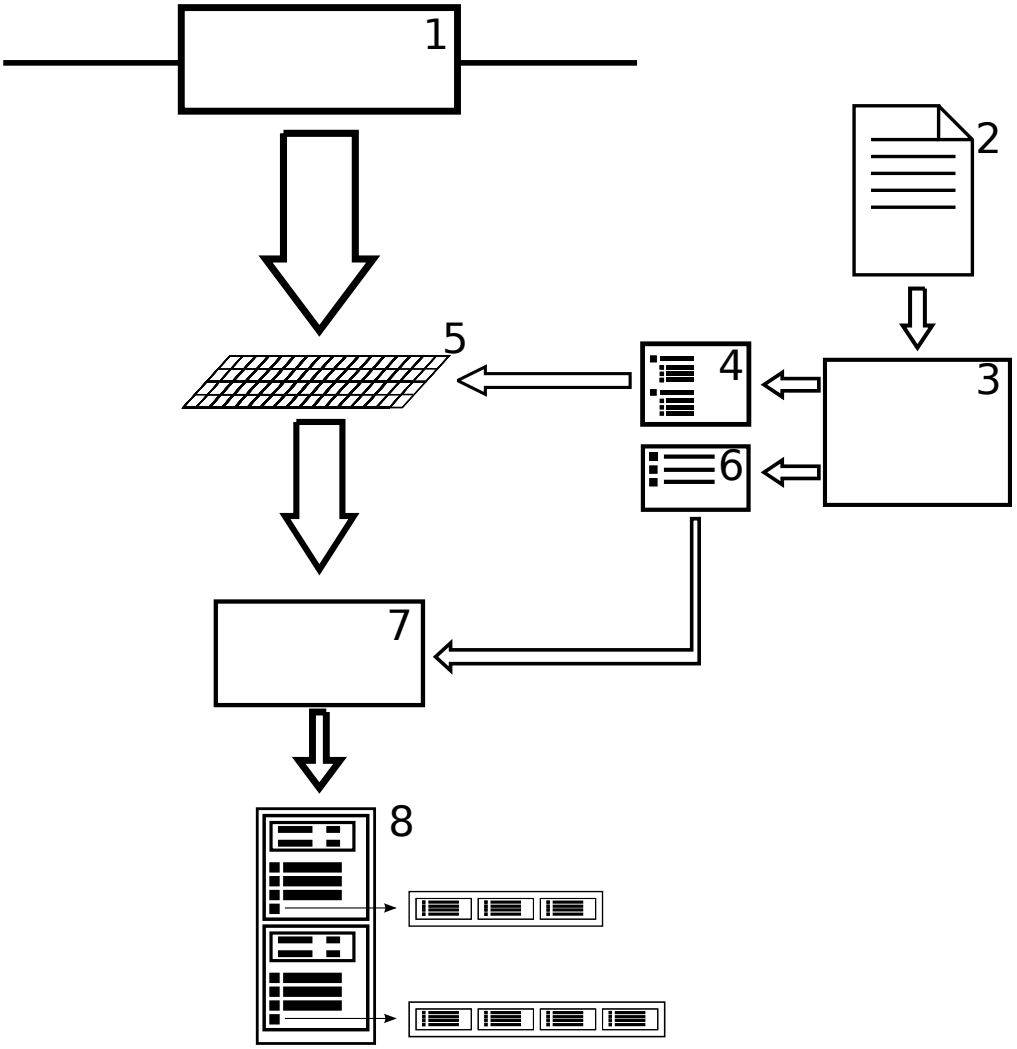
4.4.5 Zobrazovanie výsledkov

Ako riešenie pre zobrazovanie výsledkov monitorovania, som zvolil webové rozhranie z dôvodu že mi to prišlo ako najlogickejšia voľba pre zobrazovanie HTTP monitorovania, a takisto preto, lebo som mohol využiť ako úložisko databázu SQL ktorá poskytuje metódy pre základné operácie ako je zoraďovanie, výber a agregácia. V prípade že by som sa rozhodol implementovať iný druh reprezentácie ktorý by využíval iba C kód bolo by potrebné implementovať tieto metódy, čo som pokladal za náročné.

Databáza

Prvé čo bolo v súvislosti s týmto problemom potrebné riešiť, bolo zvolenie vhodného datbázového enginu a štruktúra tabuliek databázy, nato aby sme mohli exportovať dáta uložené v hashovacej tabuľke, bolo potrebné implementovať rozhranie pre komuniáciu medzi databázou a modulom.

Napokon som ako databázový engine zvolil SQLite. Dôvod pre jeho výber je ten, že jeho použitie je veľmi jednoduché, stačí iba vytvoriť vhodný súbor, netreba komplikovane konfigurovať databázu, a okrem toho tento engine bol už podporovaný na serveri *benefizio* na ktorom som implementoval a testoval modul, čiže som nemusel vlastnoručne inštalovať žiaden vlastný engine. Na druhej strane má ale obmedzenejšie možnosti než ostatné enginy, a je menej podporovaný frameworkom Django než napríklad PostgreSQL, avšak jeho funkcie mi stačili k tomu čo som požadoval od databázy.



Obrázek 4.3: Ilustrácia spracovania NetFlow záznamov, 1. vstupné rozhranie, 2. súbor s konfiguráciou, 3. parser konfiguračného súboru, 4. pravidlá filtra, 5. filter, 6. všeobecné nastavenia, 7. analyzátor, 8. tabuľka záznamov

Štruktúru databázy nebudem podrobne rozpisovať, pretože tabuľky sa snažia kopírovať relácie medzi štruktúrami a obsah jednotlivých dátových štruktúr v module, čo znamená že rovnako ako modul, aj databáza obsahuje tabuľku pre záznamy o komunikácií, ktoré sú previazané s tabuľkou s jednotlivými záznamami atď.

Pre komunikačné rozhranie som musel využiť knižnicu `sqlite3` ktorá poskytuje funkcie pomocou ktorých je možné jednoducho komunikovať s SQLite databázou. Nad touto knižnicou som naimplementoval funkciu ktorá umožňuje INSERT ľubovoľného množstva hodnôt. Keďže komunikáciu s extérnym úložiskom som od začiatku zamýšľal ako jednosmernú, nebola implementácia ďalších funkcií pre výber hodnôt z databázy potrebná.

Agregácia

V rámci znižovania pamäťovej náročnosti aplikácie bolo potrebné zaviesť určitý druh agregácie dát uložených do databázy. Nechcel som týmto úkonom zbytočne zaťažovať modul tým, že by sťahoval dáta z databázy, agregoval ich a posielal dáta naspäť. Dôvodom preto bolo to, že je to záťaž na modul a ako som spomínal, komunikácia s databázou bola od začiatku navrhovaná ako jednosmerná. Takisto som nechcel agregovať dáta predtým, ako budú zaslané do databázy, keďže som chcel aby mal užívateľ možnosť prezrieť si ešte originálne dáta. Bolo preto potrebné implementovať riešenie ktoré by bolo síce možné púšťať v prípade potreby z modulu, avšak to by bolo maximum čo by modul v rámci toho úkonu vykonal. Týmto riešením sa napokon stala SQL procedúra ktorá vezme dáta z určitého časového rozsahu, zoskupí ich podľa určitých spoločných znakov a následne ich uloží do špeciálnej tabuľky. Potom čo sú tieto dáta agregované, nemá zmysel uchovávať ich v pamäti, avšak rozhodol som sa dať užívateľovi možnosť nastaviť čas medzi agregáciou a odstránením, tento parameter je možné nastaviť v konfiguračnom súbore. Túto možnosť je možné využiť v prípade, že administrátor nekontroluje výsledky monitorovania stále, ale iba periodicky. Teraz má možnosť nastaviť že sa nezmažú zagregované dáta a bude mať možnosť si ich pozrieť.

Paralelné procesy

Keďže mnoho z operácií v module bolo potrebné opakovať periodicky (posielanie dát do databázy, agregácia) rozhodol som sa implementovať sekundárny proces ku hlavnej procesovacej slučke. Úlohou tohto procesu bolo vždy po určitej dobe spúšťať ďalšie procesy ktoré vykonali požadovanú funkciu a ukončili sa. Tento hlavný podproces, rovnako ako aj všetky jeho podprocesy, som realizoval s využitím knižnice `pthread`. Pre túto knižnicu som sa rozhodol z dôvodu, že som chcel nové procesy realizovať ako vlákna, miesto rozvetvovania, keďže bolo potrebné aby vlákna zdieľali medzi sebou dáta. To bolo nevyhnutné preto, lebo mnoho z tých procesov pracovalo aktívne s tabuľkou záznamov ktorú napĺňala procesovacia slučka.

Potom čo som mal implementovaných niekoľko paralelných procesov, rozhodol som sa spraviť posledné opatrenie pre zníženie pamäťovej náročnosti. Týmto opatrením bolo odstraňovanie neukončených záznamov po vypršaní časového limitu. Princíp spočíval v tom že som periodicky kontroloval čas posledne prijatého záznamu a v prípade že bol tento čas dávnejšie než bol časový limit, komunikácia bola považovaná za ukončenú a záznam bol odstránený. Toto malo za úlohu zbaviť sa záznamov ktoré boli neukončené buď kvôli problému na klientovej strane, napríklad výpadok internetu, alebo preto lebo klient ukončil komunikáciu, zatvorenie prehliadača. Na druhej strane je tu však možnosť že tento prístup odstráni aj veľmi dlhé prenosy dát (posielanie veľkého súboru ako jeden záznam). Preto

som dal možnosť nastaviť čas za ktorý je záznam považovaný za ukončený, tento parameter sa dá zvyčajne odhadnúť znalosťou HTTP servera.

Pre lepšiu predstavu som pridal reprezentáciu pararelných procesov a práce s databázou viď obrázok 4.4. Po tomto kroku implementáciu som mal všetky záznamy periodicky ukladané do databázy. Posledným krokom bolo teda znázorniť tieto dáta pre užívateľa.

Web rozhranie

Posledná časť mojej práce sa týkala zobrazenia nameraných a vyexportovaných dát vo forme ktorá by bola ľahko čítateľná a správne reprezentovala výkon serveru. Ako riešenie som sa rozhodol vytvoriť webovú stránku ktorá by zobrazovala grafy hodnôt uložených v databáze. Framework Django som zvolil ako základ pre svoju stránku. Tento framework pre mňa ako úplného začiatočníka bol síce náročnejší než čisté PHP, ale zato je v ňom napísaný kód podstatne kratší a ľahšie rozšíriteľnejší o ďalšiu funkcionalitu, akou boli v mojom prípade ďalšie grafy.

V tomto frameworku som napokon vytvoril funkčnú stránku s 4 grafmi a záznamami o komunikácii. Jedná sa o tieto grafy

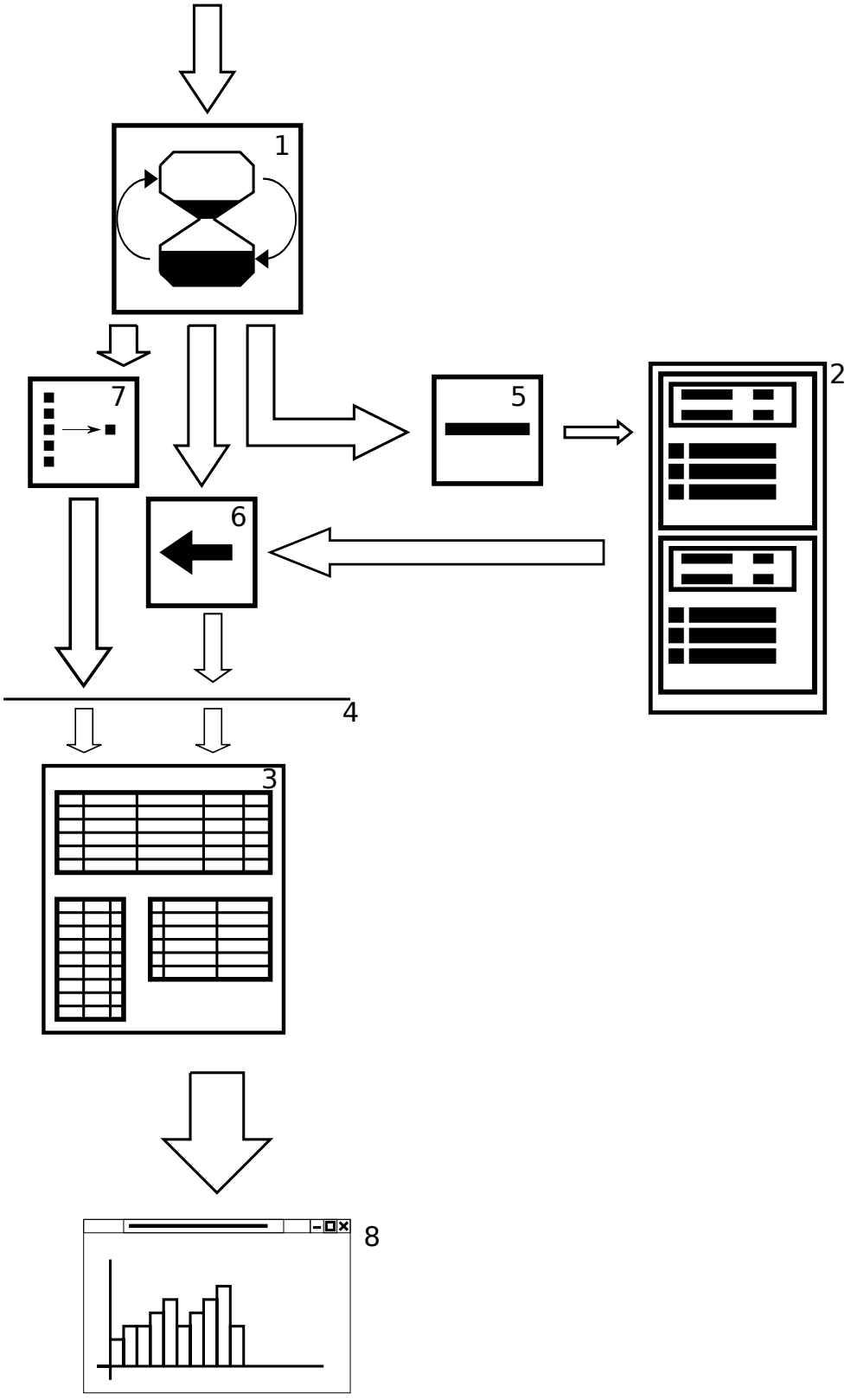
1. Dĺžka spojenia - zobrazuje neagregované dĺžky komunikácií
2. Obsah - zobrazuje oneskorenie pre jednotlivé typy obsahu
3. Priemer - ukazuje agregované hodnoty odozvy a oneskorenia
4. Počet užívateľov - počet unikátných užívateľov na stránke za určité časové obdobie

Snímky týchto grafov je možne vidieť na obrázku 4.5. Celá stránka je koncipovaná ako rozširujúce sa bloky šablón. Tento prístup je odlišný oproti bežne používanému „zahŕňaniu“(include), ktoré využívame v prípade že chceme na stránke inteligentne meniť iba niektorú časť, zvyčajne sa jedná o text, napríklad článok. V prípade zahŕňania sa k hlavnému bloku stránky, obsahu, pridajú ostatné, nie až tak často sa meniace prvky, napríklad hlavička, zápätie, menu. Prístup Djanga k tomuto problému je naopak taký, že sa zoberie celá stránka a nahradzujú sa iba časti, ktoré chceme zmeniť, v mojom prípade to bolo bočné menu a zobrazené grafy. Tým že som riešil jednotlivé časti webu takýmto spôsobom, bolo možné pridať nové položky na web veľmi jednoduchým spôsobom.

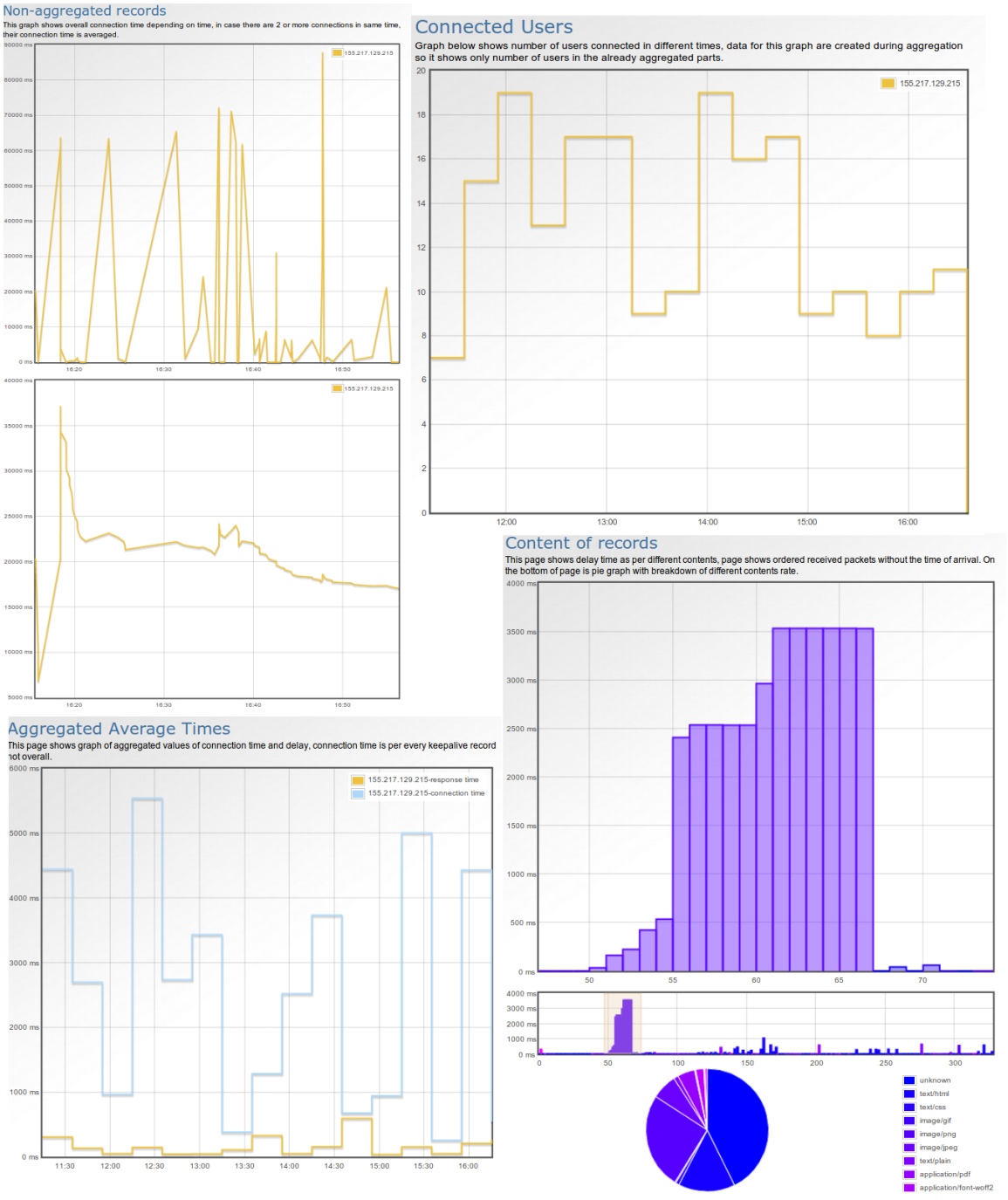
Jednotlivé grafy som vykresľoval pomocou JavaScriptovej knižnice Flot, špecializovanej na vykresľovanie grafov. Vstupné hodnoty z grafov som získaval z databázy pomocou modelu do zobrazovača, ktorý vykalkuloval hodnoty potrebné pre jednotlivé grafy a zaslal ich vo vhodnej forme na stránku. Pri každom grafe som implementoval možnosť výberu časového rozsahu pre ktorý sa má graf zobraziť a takisto IP serveru pre ktorý chceme graf vygenerovať. Zobrazenie záznamov o komunikácií je takisto implementované, jeho využitie je však skôr iba na účely nájdenia záznamu o známej komunikácii, keďže neumožňuje žiadnu rozsiahlu funkcionalitu akou by bolo filtrovanie záznamov alebo zoradzovanie.

4.5 Testovanie

Posledný bod zadania hovoril o potrebe výsledný produkt otestovať pri práci so živými dátami. So živými dátami som pracoval takmer od začiatku, čiže som vedel že modul funguje aj v takomto toku dát správne. Jediný bod ktorý som teda potreboval otestovať bolo dlhodobejšie sledovanie spojené so zmenou správania a výsledku pri rôzných konfiguráciach.



Obrázek 4.4: Ilustrácia pararelného procesu pri práci s databázou, 1. časovač, 2. tabuľka záznamov, 3. databáza, 4. komunikačné rozhranie, 5. proces mazania, 6. proces exportovania, 7. proces agregácie, 8. webové rozhranie



Obrázek 4.5: Grafy vygenerované z monitorovaných dát, dáta pochádzajú z monitorovania HTTP serveru FIT VUT, zľava zhora sa jedná o graf dĺžky spojenia, počtu užívateľov, agregovaného priemeru a obsahu

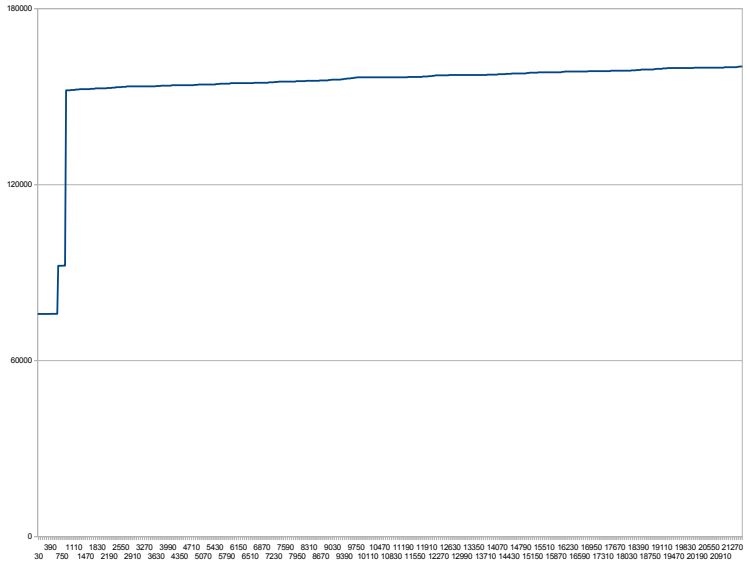
Keďže informácie o IP adresách záznamov som dostával v anonymyzovanej forme, musel som požiadať o anonymyzované tvary IP adries serverov inštitúcií na ktorých bola pravdepodobná vysoká HTTP prevádzka.

Napokon som dostal IP adresy serverov vysokých škôl FIT VUT, FI MUNI a FIT ČVUT, FEEC VUT, portálu VUT a ČVUT. Na týchto serveroch som následne pustil zároveň dva moduly monitorovania s odlišnými konfiguráciami. Konfigurácie sa zhodovali v monitorovaných serveroch aj pravidlách monitorovania pre nich. Líšili sa však v periódach mazania a exportovania dát, takisto v limite odozvy a agregačnej perióde. Od tohto merania som očakával že zobrazí rozdiely v pamäťovej náročnosti jednotlivých konfigurácií. Konfigurácie jednotlivých modulov vyzerali takto:

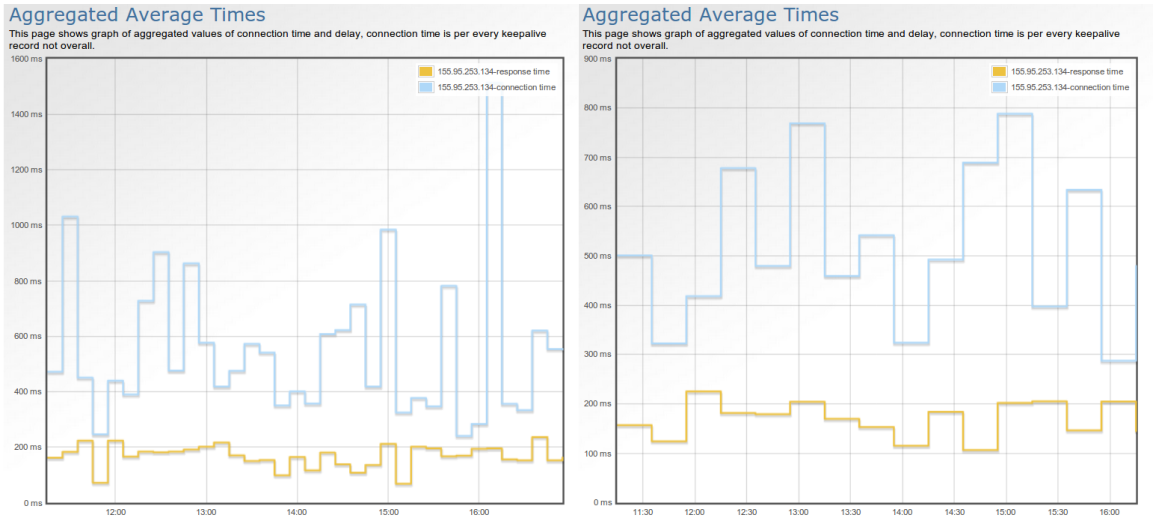
```
Explore = false;                    Explore = false;
CleanUp = true;                     CleanUp = true;
UpdatePeriod = 120;                 UpdatePeriod = 600;
RemovePeriod = 600;                 RemovePeriod = 800;
MaximumRecs = 30000;                MaximumRecs = 30000;
EmergencyPeriod = 300;              EmergencyPeriod = 300;
TimeoutPeriod = 90;                 TimeoutPeriod = 300;
EmergencyTimeoutPeriod = 1200;      EmergencyTimeoutPeriod = 1200;
AggregationPeriod = 600;            AggregationPeriod = 1200;
AggregationDelay = 120;             AggregationDelay = 120;
DeleteDelay = 0;                    DeleteDelay = 0;
```

Monitorovanie som nechal bežať skoro 5 hodín, počas ktorých modul nevykázal žiadne známky nestability, avšak ani raz počas tejto doby sa nelíšila pamäťová náročnosť procesov o viac než 10 MB. Po spustení a inicializácií modulu zaberali oba moduly 79 MB. Po 5 hodinách zaberal jeden z procesov 180 MB a druhý 160 MB virtuálnej pamäte, pričom väčšina tejto náročnosti bola spôsobená využívaním vlákien ktoré si alokovali asi 80 MB pamäte. Z toho nám vyplýva že modul si za 5 hodín práce nenaalokoval viac než o 30 MB pamäte, graf zobrazujúci zmenu pamäťovej náročnosti počas behu programu je obrázok 4.6. Tento test teda neukázal žiadne odlišnosti v závislosti od konfigurácie, zato ale ukázal na nízke pamäťové nároky modulu aj pri veľkom obsahu dát. Alokácia bola pravdepodobne spôsobená tým, že modul nemôže mazať žiadne záznamy o ktorých uložil komunikáciu do databázy, z dôvodu že si musí pamätať identitu ktoré má dané spojenie v databáze aby neduplikoval záznamy. Toto je dôsledok toho, že komunikácia s databázou je jednosmerná, modul teda nemôže zistiť opätovne identitu spojenia z databázy. Za čas svojho behu spracoval a uložil modul okolo 50 000 záznamov, avšak ani raz počas tejto doby nevystúpila záťaž týchto dvoch procesov na procesor nad 30%, z toho je možné odvodiť že modul by bol schopný spracovať omnoho väčšie množstvá dát.

Keďže som chcel porovnať výsledky týchto dvoch monitorovaní, ukladali obe moduly svoje namerané hodnoty do rôznych databáz. Na obrázku 4.7 je možné vidieť graf priemerných hodnôt, vygenerovaný z dát pre server FIT ČVUT, je na ňom možné vidieť že výsledné hodnoty sa takmer úplne zhodujú, líši sa len frekvencia agregácie. Pravdepodobne zaujímavejšie výsledky by prinieslo meranie s konfiguráciami v ktorých by boli periódy podstatne nižšie, kde by sa mohlo prejaviť nesprávne zahadzovanie záznamov kvôli vypršaniu limitu, avšak to nebolo účelom testovania pretože modul má slúžiť prevažne na dlhodobé, pomalé sledovanie.



Obrázek 4.6: Graf pamäťovej náročnosti modulu v závislosti od času



Obrázek 4.7: Porovnanie nameraných hodnôt modulmi s odlišnými konfiguráciami

Kapitola 5

Záver

Táto práca mala za úlohu demonštrovať možnosti využitia frameworku Nemea na účely, pre ktoré sa bežne nevyužíva, a to monitorovanie siete, nie kvôli detekcii útokov, ale kvôli zisťovaniu výkonnosti HTTP serverov. Kvôli tomuto cieľu bolo potrebné vytvoriť plugin, ktorý zlepšil potenciál základného zdroja dát pre systém Nemea, NetFlow exportéra. Následne bolo potrebné navrhnúť a implementovať funkcionalitu modulu ktorý mal zbierať, filtrovať a analyzovať prichádzajúce dáta. Pridať bolo potrebné aj metódy pre znižovanie veľkosti uložených dát a ukladanie dát na externé úložisko, ktorým bola databáza, kvôli čomu bolo potrebné naprogramovať rozhranie medzi dvoma odlišnými prostrediami. Napokon bolo potrebné naprogramovať webové rozhranie ktoré by umožňovalo namerané hodnoty reprezentovať.

Návrh tohto modulu ma naučil mnoho o sietiach, monitorovaní a systéme Nemea. Pri implementácií som využil vedomosti jazyka C a SQL nadobudnuté počas štúdia, k čomu som som sa musel naučiť programovanie webových aplikácií v Pythone a frameworku Django.

V prípade že by som mal možnosť pracovať na tomto projekte ďalej, rozhodne by som rozšíril spektrum grafov zobrazujúcich monitorované hodnoty o grafy zobrazujúce závislosti jednotlivých metrík od iných hodnôt ako času, napr. doba odozvy v závislosti od počtu pripojených užívateľov. Ďalej by som určite pridal obojstrannú komunikáciu modulu s databázou a tým odstránil rast pamäťovej náročnosti.

Výsledok tejto práce rozhodne ukazuje možnosti využitia frameworku Nemea za účelmi monitorovania výkonu. Táto práca ma donútila naštudovať mnoho z rôzných oblastí informatiky, som však presvedčený že tieto vedomosti ešte v budúcnosti využijem.

Literatura

[1] Performance Metrics for Websites.
URL http://community.blazemeter.com/knowledgebase/articles/34412-performance-metrics-for-websites

[2] Bartoš, V.; Žádník, M.; Čejka, T.: Nemea: Framework for stream-wise analysis of network traffic. Technická Zpráva 9/2013, CESNET, December 2013.
URL http://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf

[3] Claise, B.; Trammell, B.; Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC, 2013.
URL http://tools.ietf.org/html/rfc7011

[4] Claise, E., B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954, 2004.
URL https://tools.ietf.org/html/rfc3954

[5] Costill, A.: SEO 101: How Important is Site Speed in 2014? July 2014.
URL http://www.searchenginejournal.com/seo-101-important-site-speed-2014/111924/

[6] Dragich, L.: The Anatomy of APM – 4 Foundational Elements to a Successful Strategy.
URL http://www.apmdigest.com/the-anatomy-of-apm-4-foundational-elements-to-a-successful-strategy

[7] Fielding, R.; Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC, 2014.
URL http://tools.ietf.org/html/rfc7230

[8] Fielding, R. E.; J. Reschke, E.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Technická zpráva, 2014.
URL http://tools.ietf.org/html/rfc7230

[9] Hofstede, R.; Celeda, P.; Trammell, B.; aj.: Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *Communications Surveys Tutorials, IEEE*, ročník 16, č. 4, Fourthquarter 2014: s. 2037–2064, ISSN 1553-877X, doi:10.1109/COMST.2014.2321898.

[10] Holovaty, A.; Kaplan-Moss, J.: *The Definitive Guide to Django: Web Development Done Right*. Apress, December 2007.

[11] Janssen, C.: Website Monitoring.
URL http://www.techopedia.com/definition/29994/website-monitoring

Příloha A

Obsah CD

- \src - obsahuje všeky zdrojové súbory modulu a webu
- \doc - zdrojové texty k písomnej správe
- pisomna_sprava.pdf - preložená pisomná správa